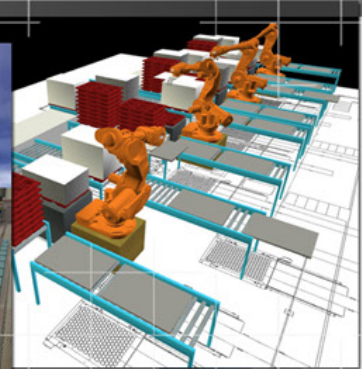
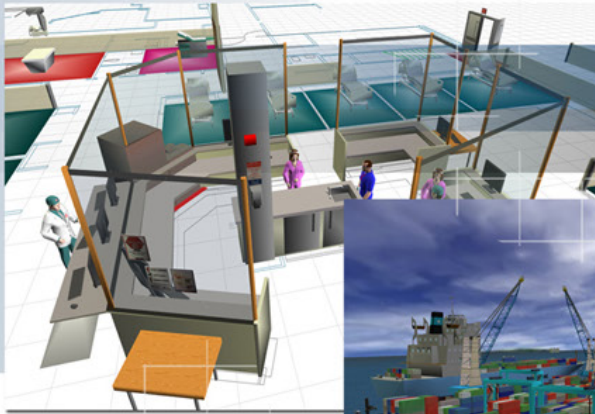
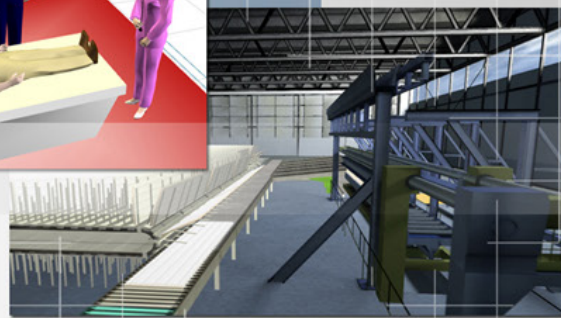


Applied Simulation

Modeling and Analysis using **FlexSim**
3D Simulation Software



Malcolm Beaverstock, PhD
Allen Greenwood, PhD, PE
Eamonn Lavery, PhD
William Nordgren, MS CIM



visualizing and optimizing dynamic systems

Applied Simulation

Applied Simulation

Modeling and Analysis using *FlexSim*

Third Edition

Compatible with FlexSim 6.0

Malcolm Beaverstock, PhD

Allen Greenwood, PhD, PE

Eamonn Lavery, PhD

William Nordgren, MS CIM

Exercises in the book require either the Student version or Standard version of *FlexSim*

However, the free, evaluation version of *FlexSim* can be used for exercises through Chapter 6.

Model files for the book are available in the BookDownloadFiles.zip file on the FlexSim website

www.flexsim.com

© 2011 FlexSim Software Products, Inc.

Canyon Park Technology Center

Building A – Suite 2300

Orem, Utah 84097 USA

Phone (801) 224-6914 • Fax (801) 224-6984

sales@flexsim.com

Applied Simulation Modeling and Analysis using FlexSim

by Malcolm Beaverstock, PhD
Allen Greenwood, PhD, PE
Eamonn Lavery, PhD
William Nordgren, MS CIM

Copyright © 2012 FlexSim Software Products, Inc. All rights reserved.
Printed in the United States of America

Published by FlexSim Software Products, Inc., Canyon Park Technology Center, Building A Suite 2300, Orem, UT 84097 USA.

FlexSim software and books may be purchased for educational, business, or sales promotional use. For more information, please contact our sales department: +1-801-224-2914 or sales@flexsim.com.

Editing: Shanna Warr, Katherine Bobo

Production Editing: Shanna Warr, Katherine Bobo, Authors

Cover Design: Stacy Geisberger, Kris Geisberger, and Ben Wilson, with special thanks to Olivier Pellegrin, Brenton King, TALUMIS, and others who provided screen captures of actual *FlexSim* simulation models.

Printing History:

January 2011:	First Edition, version 1 (pre-release)
August 2011:	First Edition, version 2 (pre-release)
September 2011:	First Edition, version 3
January 2012:	Second Edition, version 1
July 2012:	Third Edition

The *FlexSim* logo is a registered trademark of FlexSim Software Products, Inc. Other registered trademarks belonging to third parties are used within this work. Where those designations appear in this book, and FlexSim Software Products, Inc. was aware of a claim, the designations have been printed in italics, caps, or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained here.

ISBN: 978-0-9832319-2-9

To Janet and Jane – for your patience and support

Acknowledgements

Many people have contributed their knowledge and experience to the material in this book. The authors wish to thank everyone in the FlexSim main office for their support and contributions. We especially extend our gratitude to those individuals at General Mills, Inc. and the Center for Advanced Vehicular Systems–Extension at Mississippi State University who encouraged and supported the use of simulation.

Preface

Simulation is an applied technology that is especially useful for analyzing and solving problems. Applying simulation begins by being clear on the problem definition, the reasons for simulating, and the expected outcomes. Simulation with no objective is counterproductive.

A person using simulation then must balance their understanding of the problem with their knowledge of the details of simulation: the underlying simulation concepts, application software, and the analysis methodologies that are employed. Consequently, the most effective way to learn how to successfully use simulation is through learning how to apply it.

A major challenge in teaching applied simulation is the question of how to effectively blend and balance an understanding of fundamental principles and concepts with the practical side of building simulations. The intent of this book is to help bridge that gap and improve the effectiveness of simulation courses. Not all readers of this book will become simulation experts, but hopefully they will want to utilize the technology to help them or others make better decisions. Consequently the material takes the reader through three levels of users: Occasional, Intermediate, and Advanced.

FlexSim was chosen for use with this text because of its ease-of-use and rich functionality that allows users to focus on simulation concepts and methods. This is not intended to be a *FlexSim* manual as the *FlexSim* Help files and tutorials are more than adequate for that purpose.

The opening chapters focus on Occasional Users and provide a base for all user levels. The chapters establish the professional practice of applied simulation: the basics, economic justification, when simulation is needed, and a methodology for defining a simulation project. The intent is to demonstrate how simulation modeling and analysis is used to understand and resolve practical problems.

The next section in the book deals with Intermediate Users, those who desire to build simulations, but do so infrequently. These chapters focus on the basics of simulation software, statistics, equipment reliability, designing experiments, and model development. Simulation software details, if desired, are available in the appendices or from the *FlexSim* Help system.

Chapters for Advanced Users introduce topics such as writing custom logic, dealing with production schedules, and simulating fluid flow. Later chapters include a discussion of simulation software architecture along with examples of more advanced applications such as distributed simulation and agent-based simulation. Appendices cover *FlexSim* application details, application notes for each exercise, and specialized application topics.

Contents

Chapter 1 Simulation as a Tool for Understanding	1
Section 1-1 What is simulation?	1
Section 1-2 Working in a global, dynamic environment	2
Section 1-3 Simulation in everyday life	7
Section 1-4 Where is the money?	10
Section 1-5 Simulation users	13
Chapter 2 Simulation Applications	21
Section 2-1 Modeling and simulation	21
Section 2-2 When to use simulation	25
Section 2-3 The simulation software marketplace	33
Section 2-4 Simulation and other tools	37
Chapter 3 Using Simulation to Solve Problems	45
Section 3-1 Using a simulation	45
Section 3-2 Simulations with random events	46
Section 3-3 Examples and exercises	47
Exercise 3-1 Coasting around	49
Exercise 3-2 Farm Pride	51
Exercise 3-3 Martian Transfer Station	53
Exercise 3-4 Slime Inc.	55
Exercise 3-5 Danson Electronics	57
Exercise 3-6 Grandma's Pie Emporium	58
Section 3-4 The next step	60
Chapter 4 Professional Practice of Simulation	63
Section 4-1 Confidence	63
Section 4-2 The Simulation Modeling and Analysis (SMA) Life Cycle	65
Section 4-3 The modeling and analysis process	66
Section 4-4 Roles in a SMA project	68
Section 4-5 Model-based decision-support systems	70
Section 4-6 Simulation modeling and analysis success factors	72
Chapter 5 Managing a Simulation Project	75
Section 5-1 The starting point	75
Section 5-2 Define the system and the reason	77
Section 5-3 Prepare a flow diagram	81
Section 5-4 Initial sizing of resources	87
Section 5-5 Building the simulation	88

Section 5-6 Results and analysis	90
Section 5-7 Example of a project template	92
Exercise 5-1 Fister's Frozen Foods	97
Chapter 6 Building Basic Simulation Models	105
Section 6-1 Simulation environment.	106
Section 6-2 Simulation components	109
Section 6-3 Fixed resources.	110
Section 6-4 Transporting items	112
Section 6-5 Editing objects	114
Section 6-6 Making connections	118
Section 6-7 Moving flowitems.	120
Section 6-8 Creating simple learning models	122
Section 6-9 Single-run statistics	123
Exercise 6-1 Johnson Pharmaceutical	125
Exercise 6-2 Lucky Air.	127
Chapter 7 Adding Model Logic and Managing Data	131
Section 7-1 Assigning attributes to objects	131
Section 7-2 Adding logic	132
Exercise 7-1 More Lucky Air	138
Section 7-3 Managing data tables	139
Exercise 7-2 Even more Lucky Air	140
Chapter 8 Managing Entities and Time Tables	143
Section 8-1 Grouping and ungrouping flowitems	143
Exercise 8-1 Hampton International.	146
Section 8-2 Mobile resource objects (task executers).	149
Section 8-3 Displaying information on the screen	154
Section 8-4 Establishing time tables	155
Exercise 8-2 Steve's Stone Cutting.	156
Exercise 8-3 The Crafty Framer.	158
Chapter 9 Modeling Randomness	163
Section 9-1 Data-driven probability distribution selection	163
Section 9-2 Selecting probability distributions in the absence of data	166
Section 9-3 Use of probability distributions in simulation	168
Section 9-4 Sampling from continuous random variates.	169
Section 9-5 Sampling from discrete random variates.	172
Section 9-6 Generating random numbers.	173
Section 9-7 Putting it all together	175

Section 9-8	Using random samples to drive a simulation	176
Section 9-9	Reducing variability in the samples	182
Chapter 10	Analyzing Simulation Output	187
Section 10-1	Key tactical experimental design parameters	188
Section 10-2	Terminating versus non-terminating systems	192
Section 10-3	Determining the number of replications	195
Section 10-4	Creating experiments using <i>FlexSim's</i> Experimenter	196
Section 10-5	Comparing two alternatives	198
Section 10-6	Variance reduction through common random numbers	200
Section 10-7	Multiple comparisons of alternatives	201
Chapter 11	Including Reliability in a Simulation.	205
Section 11-1	Performance	205
Section 11-2	Reliability	206
Section 11-3	Estimating MTBF and MTTR.	208
Section 11-4	Simulating machine failures	210
Section 11-5	Setting MTBF and MTTR in a simulation	212
Section 11-6	Using personnel for repairs	215
Section 11-7	Surge	216
Exercise 11-1	Keggler's Brew.	217
Exercise 11-2	Chairs for Tots.	220
Chapter 12	Customizing Model Logic	227
Section 12-1	Hierarchical software architecture	227
Section 12-2	Understanding how objects work	231
Section 12-3	Scripting basics	233
Section 12-4	Creating custom logic	238
Section 12-5	Custom logic example	244
Exercise 12-1	Hilltop Steel Works	248
Chapter 13	Communicating Among Objects.	251
Section 13-1	Communicating between objects.	251
Section 13-2	Label Tables	256
Section 13-3	The Recorder	257
Section 13-4	Customizing object placements.	258
Section 13-5	Reusing custom objects	260
Exercise 13-1	Fister's Express	261
Exercise 13-2	Peoples Surgery Center.	264
Chapter 14	Simulating Fluid Flow	269

Section 14-1 Basics of fluid-flow simulation	269
Section 14-2 Fluid objects	271
Section 14-3 Specifying Fluid Operations	272
Exercise 14-1 James Peanuts	275
Exercise 14-2 Western Grain	278
Chapter 15 Simulating Production Schedules	283
Section 15-1 Controlling production lines	283
Section 15-2 System controller	284
Section 15-3 Line controller	286
Exercise 15-1 Custom Shapes, Inc.	289
Appendix	295
Appendix for Chapter 3	297
Section 1 General toolbars	297
Section 2 Viewing the simulation using a mouse	299
Appendix for Chapter 5	301
Section 1 Simulation Project Template	301
Section 2 Fister's Foods exercise	305
Appendix for Chapter 6	309
Section 1 Modifying the basic View settings	309
Section 2 Selecting and modifying object visuals	310
Section 3 Picklists	312
Section 4 Conveyor layout	313
Section 5 Exercise 6-1 Johnson Pharmaceutical	315
Section 6 Exercise 6-2 Lucky Air	316
Appendix for Chapter 7	323
Section 1 Putting labels on flowitems	323
Section 2 Modifying trigger logic with picklists	324
Section 3 Example 7-1 More Lucky Air	326
Section 4 Exercise 7-2 Even More Lucky Air	332
Appendix for Chapter 8	335
Section 1 Exercise 8-1 Hampton International	335
Section 2 Connecting and calling task executers	336
Section 3 Configuring timetables	340
Section 4 Exercise 8-2 Steve's Stone Cutting	342
Section 5 Exercise 8-3 The Crafty Framer	343

Appendix for Chapter 9	.345
Section 1 Notes for Using <i>ExpertFit</i>	345
Section 2 Statistical distributions in <i>FlexSim</i>	346
Section 3 Tables of normal and t distributions	351
Appendix for Chapter 10	.355
Section 1 Setting up the Experimenter	355
Section 2 Defining scenarios	357
Section 3 Advanced experimenter options	358
Appendix for Chapter 11	.359
Section 1 Exercise 11-1 Kegglers Brew	359
Section 2 Exercise 11-2 Chairs for Tots	360
Appendix for Chapter 12	.363
Section 1 Commonly-used commands	363
Section 2 FlexSim software architecture	367
Section 3 Exercise 12-1 Hilltop Steel	371
Appendix for Chapter 13	.375
Section 1 Recorder details	375
Section 2 Exercise 13-1 Fister's Express	376
Section 3 Exercise 13-2 Peoples Surgery Center	379
Appendix for Chapter 14	.383
Section 1 Exercise 14-1 James Peanuts	383
Section 2 Exercise 14-2 Western Grain	385
Appendix for Chapter 15	.387
Section 1 Custom Shapes Exercise	387
Appendix - Interacting with Other Applications	.391
Section 1 Microsoft Excel import/export	391
Section 2 Importing AutoCAD drawings	392
Appendix - Advanced Techniques	.395
Section 1 Task Sequences	395
Section 2 Creating AVI files to document the simulation	397
Section 3 Creating custom GUIs	398
Section 4 Kinematics	402
Section 5 Reports, statistics, and documentation	403
Appendix - Related Applications	.407

Section 1	Health Care	407
Section 2	Container Terminal (CT).....	408
Section 3	Express It 3D presentation software	409
About the Authors.....		411

Chapter

1

Simulation as a Tool for Understanding

This first chapter sets the foundation for studying simulation as an applied technology and analyzing and solving problems that exist in real systems. It provides the context for the book by defining what simulation is and the characteristics of the systems in which simulation is applied. These systems are widespread and diverse, including all areas of manufacturing and mining, as well as service operations such as healthcare, security, transportation, retail, and distribution, to name a few. The chapter introduces why simulation is used and what value it adds to all facets of decision making. It also describes the capabilities and skills of the three types of simulation users – Occasional, Intermediate, and Advanced. The book is organized into three main sections that address the needs of these three types of users.

Section 1-1 What is simulation?

As the title indicates, this book is about simulation with an emphasis on applied, real life situations. The primary focus will be on modeling and analysis. Before starting the book's journey the term *simulation*, as it is used in this text, should be defined.

Generally, the verb *to simulate* means to mimic or imitate. This book takes a broader view of the verb as it relates to actual applications. Here, *to simulate* means to mimic or imitate through experimentation with a model (or representation) of some real system; however, simulation also involves more than just mimicking or experimenting. It involves performing such activities as defining, designing, and constructing a model or representation; defining the experiments to be conducted; collecting and analyzing data to drive the model; and analyzing and interpreting the results obtained from the experiments. Therefore *to simulate*, as referred to in this book, means to partake in a process that encompasses all of the above activities.

When a process is simulated, the result is often referred to as a simulation. The complete definition of the term *simulation*, as used in this book, refers to the holistic act that includes a variety of activities (definition, design, construction, analysis, interpretation) and not to just a single experiment or execution of a model on a computer.

Section 1-2 Working in a global, dynamic environment

Economics, in the form of money, land, power, or control, is the primary motivator that influences the way people live. All life is influenced by the shared basic elements of air, water, and land. Communications and computer technology have brought our thought processes and actions even closer together. Actions in one area now impact others as complex, and often non-intuitive, ripple effects. These ripple effects can be short or long term in nature. Even if everyone in the world believed they were working for the same objectives of peace and justice, actions by one segment could unintentionally contradict what another was doing.

Competition, risk, and innovation

Organizations, businesses, and individuals worldwide search for ways to lower costs and optimize their use of critical resources; however, achieving this in a changing, integrated, and dynamic worldwide environment becomes a major challenge. In recent history, quality circles brought people together to avoid waste and improve their products. Today, organizations search for “lean” systems that simplify their operations. They prepare value stream maps to identify wasted time and effort. Optimization is considered a key to success. But in an ever increasing dynamic and integrated world, even these efforts can quickly become out of date or fall prey to unintended consequences.

Success in the world economy is often viewed in terms of competition, risk, and innovation. Business success involves speed and leverage in making decisions. In a time dependent world it is critical to plan, execute, and work efficiently the first time, every time. The changing demand for goods and services is just one example of the need to make timely decisions.

Consider the implications of competition and risk for consumer based operations. Providing goods and services without direct demand information is not economically viable. The result is goods sitting in warehouses dragging down corporate profits. Electronic goods sit idly while technology passes them by, food reaches its expiration date sitting on the shelves, highly paid service providers sit with no income stream waiting for clients.

Just as manufacturers can't afford to keep large reserves on hand, they also can't afford to run out of product and lose customers to their competitors. Sellers of goods, such as large retailers, refuse to accept warehousing costs for products that they might have to sell at “clearance” prices. In response to the need for speed and

**Competition,
risk, and
innovation are
keys to success
in a global
economy**

**Balancing
supply and
demand is
critical to
maximize
profitability**

accuracy in meeting consumer demands, a philosophy of *pull demand* emerged. In this scenario, goods and services are produced just as they are needed. While this sounds good in theory, it is difficult to successfully implement.

Very similar scenarios apply to other areas. Banks want to optimize their staffing and have the correct number of lines open for various levels of customer demand; grocery stores have a similar problem balancing check-out lines with personnel loading shelves. Military services have to keep spare parts for equipment and send them to where they're needed. Hospitals need to have the correct number of rooms, equipment, and staff available for various patient levels, yet minimize their operating costs.

Innovation is another important key to success as it provides a competitive or business advantage for one company over another. Innovation comes in many forms. It can be a great new idea for a product, a new machine that improves the manufacturing cost structure, or a collection of small improvements that are built along the way. People in organizations are learning concepts such as “Lean”, “Six Sigma”, and “Continuous Improvement” to analyze and improve their operations and the way they interact while doing business. Many of the concepts involve people at all levels of an organization studying and improving their operations using simple analytical tools.

Unfortunately, a company or organization generates more innovative ideas than it can handle. Priorities for funds have to be considered as well as the impact of the innovative concept. With highly interactive systems, a great idea for one area may actually harm another. For example, a lean team on the production floor gave a recommendation for how they could increase the capacity of their assembly line without the need for additional equipment or personnel. They suggested that by making a minor change in the resource flow path, they could share part of the path with a nearby line that had excess capacity. There also were proposals from lean teams on another line claiming efficiency improvement, including the line sharing part production. Someone has to decide if the proposed efficiency gain would actually be achieved and be sustainable through production forecasts for both lines.

While the previous example may not have involved a lot of capital cost to implement, the results of making the right choice were significant. When a major capital project is considered (new plant, new line, major expansion, etc), time, money, and business impact are all critical. The major question is *for all that's being invested will we get what we think we'll get and when will we get it*. The worst possible outcome is an “end of project mismatch discovery,” in which people suddenly realize that the project outcome doesn't match what was planned.

Major projects normally consist of various phases with at least two major decision points for commitment of funds (Figure 1.1). The cumulative amount of cash expended rises

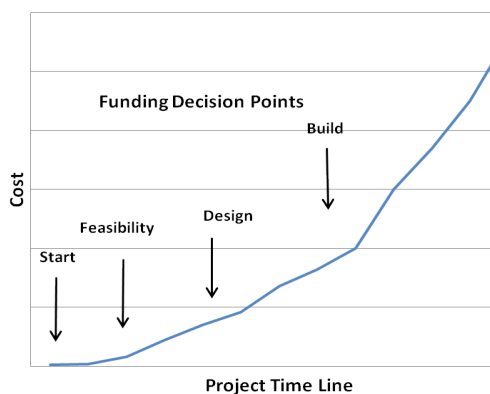


Figure 1.1 Capital project costs

The impact of decisions is often based on when they are made

exponentially as the project continues, so the earlier a serious problem in concept or design can be detected, the lower the sunk cost will be. The cost of correcting or mitigating any problem is also lower during early project phases; thus, there is a definite need to be correct the first time through a project.

For example, a large paint company thought of a major expansion to produce a new, environmentally friendly, long-lasting paint. The line would use a new manufacturing concept for producing the paint. Projected demand for the various paint grades and colors appeared strong, so flexibility in the line would be important. Before proceeding, they had to study various proposals to find the most cost-efficient one that produced the results they needed.

An emergency response organization for a major city worried about its capability to handle various situations. At issue were the number of trained personnel required, the amount and positioning of vehicles and supplies, and the city resources available for immediate and long term operations.

Systems and complexity

Being able to understand and analyze the situations just described requires what is known as systems analysis. The basic definition of a system is

a set of related elements within some stated boundary.

In practice, a system can be as big as an entire plant or organization or as small as a work cell, hospital emergency room, or bank teller line. What is of interest is the behavior of a system as a whole both internally and with its environment. That behavior is distinct from the behavior of any of the individual system components. Four general types of systems have been identified by Checkland² and described by Robinson¹:

- *Natural systems*: Weather and galaxies, etc.
- *Human activity systems*: Call centers, train stations, etc.
- *Designed physical systems*: Buildings, equipment, transportation vehicles, etc.
- *Designed abstract systems*: Mathematics, music, literature, etc.

This book focuses on the interaction of human activity systems and designed physical systems, often referred to as operations systems. These systems are normally involved with the production of goods and services, such as manufacturing processes, service delivery, and transportation and logistics operations.

All decision makers throughout an organization have to try to figure out what's happening—or going to happen—in their systems and how to take the appropriate action. In essence, they try to understand the consequences of possible actions prior to taking action. Decision making is easy for simple systems and when there are no options to choose from; however, operations systems are not easy to understand and offer a large number of possible options or actions.

Trying to analyze and make the right decision is difficult for any single person because each system has one or more of the following behavioral characteristics:

1. Components within a system are subject to their own random events
2. Random events in the environment impact the system
3. System behavior is time dependent
4. System components have complex interactions, and thus there are many connecting paths within a system

Random events are common in operations systems; some examples of random events would be orders for production based on real-time sales, customer arrival times and their particular needs, machine breakdowns, response time of operators to answer an event, arrival time of material, and material losses. When multiple pieces of equipment are involved and operations are integrated, these random events all compete with each other to impact overall performance.

Decision making

When a decision maker attempts to analyze a system and formulate a plan for optimizing its performance, that person may well find such an endeavor extremely difficult. Simply thinking through such issues using simple calculations is no longer viable because of dynamics and the random nature of the involved systems. In today's business and economic environment, the only certainty we know is that nothing is certain. Change will happen at an even faster rate.

As a result, methods were developed to help a decision maker analyze systems, these methods became known as decision support systems. A decision support system acts as a “window” or analysis tool to help the decision maker formulate action plans. Simulation is one such application used as a decision support tool. A simple and concise definition of using simulation in a decision support role is

experimentation with a simplified imitation (on a computer) of an operations system as it progresses through time, for the purpose of better understanding and/or improving the system.¹

Simulation in a decision support role is illustrated in Figure 1.2. In any type of system, decision makers need to understand, analyze, design, and manage the system; all of these activities involve making decisions. The decisions typically involve real systems that are comprised of complex processes that have highly interdependent components, which exhibit variability and change over time. Without the use of a model, decision makers will use a variety of methods to gather information from a real system, processes it, and then choose an action that causes a change in the system. Model-based decision making in general, and simulation-based decision making in particular, provide a means for decision makers to investigate a representation of the system, experiment with alternatives, and predict the effect of proposed

Random events and other behavior complicate decision making

Simulation can play a major role in decision making

changes, all external to the real system. This approach greatly increases the decision space (allows many more alternatives to be evaluated), is not intrusive on the real system, and enables risk of the actions to be assessed. In order for the decision maker to more effectively interact with simulation models directly, models need to be embedded in decision-support systems that facilitate entering data into the model and enhance the presentation of output from the model

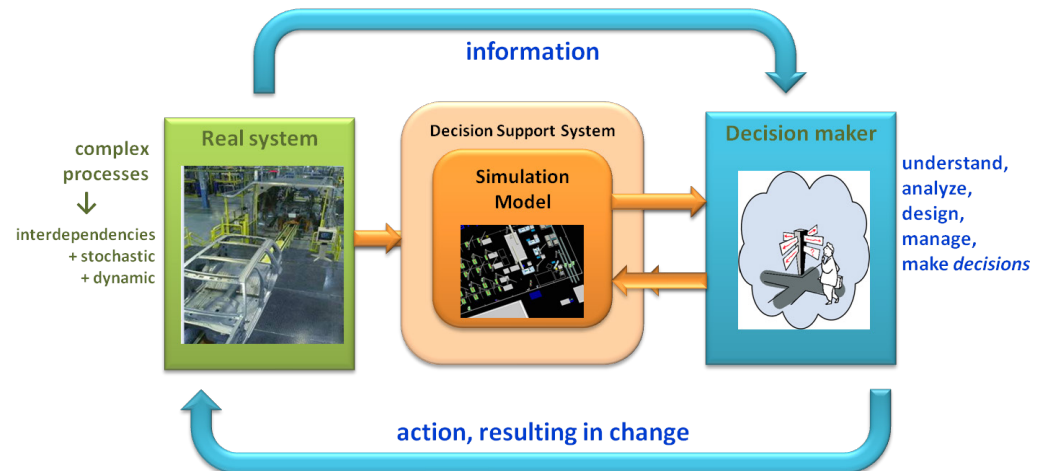


Figure 1.2 Model-based decision support system

Simulation can directly identify areas of opportunity for cost savings or efficiency; it can also play a major role in risk analysis. Simulation provides insight into risk analysis by increasing knowledge of system's dynamics and interactions. It also identifies the possible scenarios resulting from various actions. For example, while simulation may not be able to predict specific demands for future goods and services, it can be used to assess the impact of demand variability on a system's ability to react.

All the requirements for understanding dynamic systems have placed a new emphasis on simulation technologies and applications. People across an entire organization can use simulation to help make good decisions.

Simulation has become a common analysis tool to

- analyze systems for improvement;
- match project scope with business requirements;
- share information across organizations;
- focus on the impact of complex dynamics.

To accommodate these goals, simulation has undergone a change—a change made possible by computer and software technologies. Traditionally simulation has been associated with “experts” in the field who are entrusted with providing answers and are the repository for dynamic knowledge. Such knowledge and analytical tools now need to be available to everyone as a basic tool. The result is a change in the paradigm for those who use simulation.

Driven by technology and user needs, the way simulation is used has changed dramatically

- Traditional paradigm—limited base of users
 - Complex tools with no commonality
 - Experts receive training and use the simulation application
 - Experts solve problems for others
 - Simulation service is expanded by adding experts
- New paradigm
 - User friendly tools to empower people
 - Accommodation for users with varying levels of simulation knowledge
 - Common databases to share information
 - Expanded service through an increased user base

As a result, simulation is currently at work in a wide variety of applications for systems that are part of everyday life.

Section 1-3 Simulation in everyday life

Everything experiences change in some way, either in time, space, or interactions. The ability to understand, analyze, and even predict these dynamics has been the goal since the beginning of time. Simulation is a tool that has the ability to capture the dynamic characteristics of a system. Through simulation we can replicate historical dynamics, study them, and project possible future results.

Simulation is commonplace today and is often taken for granted. Simulation has been the mainstay of training systems for many years. Commercial pilots work countless hours on flight simulators before transporting the public. Almost every section of the military depends on simulators to train personnel and keep them ready for any eventuality. Operators of equipment, from supertankers to subway trains, use simulation to hone their skills for handling dynamic situations that may occur.

Manufacturing has been a prime area for simulation efforts. When a single artisan was working in a shop, the output was very predictable. The pace was that of the individual. The only issues may have been the procurement of raw materials and the condition of the tools. With the start of the industrial revolution, and especially the assembly line, manufacturing changed into a complex entity involving multiple pieces of equipment, operations, and people—all of which interacted with each other. Today, manufacturing depends on simulation to design and test new manufacturing concepts, prioritize changes to existing operations, and ensure that the outcome of a proposed project will meet expectations.

Consider a company's effort to successfully execute a major project. In many cases, simulation has become an integral part of design and an effective means to manage

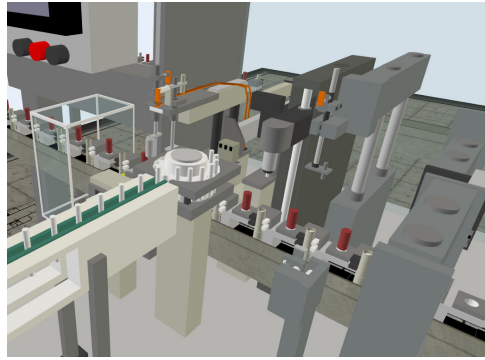


Figure 1.3 Manufacturing system design

more detailed plan, design, and cost estimate, more detail is added to the simulation. The simulation oftentimes becomes a common ground for discussions with the sales and marketing groups to get a better definition on the proposed demand.

Once a project is approved, simulation can be used to monitor and follow design decisions and identify potential problems. Again, the simulation provides a common ground for business units and project teams to negotiate changes. Staffing details are also simulated and operational details are developed with plant personnel. During the execution phase, more detail is added to the simulation in order to address scheduling and fine tune operations, all based on the latest marketing input.

Simulation beyond manufacturing

Simulation is also used beyond manufacturing processes. The trucks or trains that transport raw materials or finished products are planned and scheduled based on simulations—as are the warehouses that they interact with. Traffic patterns, including intersection light sequencing, are now simulated in order to ease congestion. Large retailers simulate just-in-time scenarios to minimize inventory yet keep shelves stocked. Railroads have long used simulations to optimize freight train length against schedule, terrain, and engine operating cost. Most recently, container terminal operations are being simulated to optimize the loading and unloading of large ships.

Simulations are being used to benefit service operations as well. Banks, ticket counters, amusement parks, cruise line check-in stations, and even airport security areas use simulation to anticipate the impact of equipment problems,



Figure 1.4 Simulations optimize container port operations

**Simulation
isn't just for
manufacturing**

sudden surges of people, or the number of open lanes. Today, simulation is helping to design more efficient health care facilities. Clinics and operating areas are prime targets to improve service quality and lower costs.

The weather patterns, including hurricane paths, are generated by simulations based on atmospheric conditions and other meteorological data. Some re-creations of accident scenes and other events are merely animations while others are actually based on the physics involved.

The most obvious use of simulation is in the ever growing field of computer games. In the early versions of the “pong” or “tennis” games, a simulated ball went back and forth between players with only the basic interaction parameters included. The ball would bounce from the paddle based on the angle of impact. At the time it was considered unique. Since then game simulations have advanced to the stage where players are totally immersed in a virtual environment. An important side effect of the game simulation technology is that dedicated players expect to pick up any new game and start playing without having to go through extensive documentation or training.

These are but a few of the many areas where simulation technology is being used—either in a very obvious way or behind the scenes. The common thread in all the cases is that simulations are created based on some level of knowledge. That knowledge may have been experience, physical laws, historical data, sequence-of-event tables, cause-effect relationships, or, in the case of the games, a created set of rules.



Figure 1.5 Health care is an emerging area for simulation

Basic characteristics of simulations

The notion that simulations are knowledge based is the first and most important tenet:

simulations require diverse knowledge of the system being considered. That knowledge includes understanding historical, current, and/or anticipated characteristics of the system being considered. The simulation then becomes a repository of that knowledge and the resulting dynamics; and, it makes these available to others.

**Simulations
are knowledge
based.**

All simulations share two basic characteristics: they require knowledge and they provide knowledge

The use of simulation can be justified based on experience

Using simulation for early decision making (in the feasibility phase) can result in significant savings

The value of a simulation, based on current or historical characteristics, is that it represents the dynamics of the simulated system and therefore allows study, investigation, experimentation, and modification without disturbing the actual system. Alternatively, a simulation based on anticipated or planned characteristics represents the dynamic behavior of a system that doesn't presently exist and enables it to be studied and investigated prior to its existence.

The second tenet is that

simulation users gain knowledge and value from the simulation.

This simple statement means that those who look to others to simulate, analyze, and provide answers learn very little about the simulated operation and hence their own environment. To use and to help build a simulation brings a person in direct contact with the simulated environment. In doing so, a person develops a deeper understanding of how the real world system operates and behaves.

Section 1-4 Where is the money ?

Simulation is used for a reason—it has to provide a value-added return for the people and the financial resources it uses. Some of that payback is in hard cash while the rest is in soft, intangible benefits. Defined savings helps everyone—especially accountants—while the intangible benefits are appreciated by management. The examples used in this section are based on more than a thousand simulation projects over a wide variety of applications used in various business and service sectors.

As shown in Figure 1.6, benefits of simulation normally fall into four categories:

feasibility assessment, cost avoidance, detail design, and operations. Each category has both tangible and intangible benefits associated with it. As with most characterizations, there is always some overlap between the categories. The categories also line up with normal procedures for evaluating capital projects or any change to existing operations.

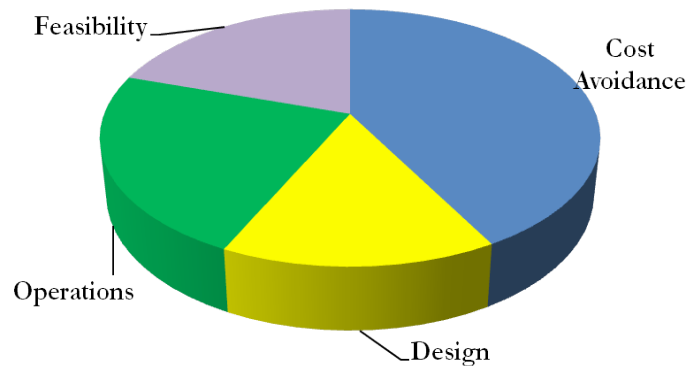


Figure 1.6 Simulation benefit categories

Feasibility assessment

In the feasibility phase of a project, new concepts are thought of and plans are made to execute them. It is the optimal time to make changes and identify possible

problems since only a small amount of effort has been expended. Typical savings during this phase have ranged from 20 to 40% of the total cost incurred during this phase. Direct savings have accrued from reducing the time to obtain an accurate estimate of the cost and benefits of a proposed project. With simulation, the time to consider and prioritize alternative recommendations is also reduced. Time savings can be directly calculated from employee cost/hr and the value of a faster time to market from business projections.

During the feasibility phase, the value of having a more accurate cost and benefit analysis carries a “soft” return by quickly focusing on those alternatives with the highest return on investment at the lowest capital costs. Such improved decision making before a project is committed is known to minimize delays and false starts later. Simulation also makes it easier to involve more people early in the project cycle, thus increasing the amount of “buy-in” from various stakeholders. It also provides a sounding board for innovative concepts and modifications that can change a project from one that provides a slight improvement to one that makes a significant contribution.

- Reduced time for accurate estimates
- Faster Lean analysis
- Improved decision making before project is started
- Easier consensus building
- Early identification of possible problems and their solutions
- Sounding board for innovative ideas

Figure 1.7 Feasibility phase savings (20-40%)

Cost avoidance

“Cost Avoidance” is a topic that accountants hate but management appreciates. This phase normally occurs during the time between the feasibility and final design. It represents funds that were once thought to be needed but no longer are and thus are available for use elsewhere. The dollars are real and often major. They can also be calculated directly.

- Eliminating non-productive equipment
- Optimizing equipment size
- Minimizing surge and floor space
- Balancing asset utilization and flexibility

Figure 1.8 Cost avoidance (2-40 % of project costs)

A major consumer products manufacturing company was able to save nearly 2 million dollars when simulation showed that a plant layout could provide the necessary production with one less packing line. A drink provider was able to reduce a project by \$500,000 by not having to provide additional raw material storage for a planned expansion. Over \$300,000 was saved in equipment and floor space when

Discovering what is or is not required is critical to the effective use of capital

smaller sized equipment was shown to be adequate. Such savings have been 2 to 40 % of the original project cost estimate.

Less tangible benefits from the use of simulation at this phase include more closely matching equipment and operations with business needs. Simulation forces marketing, engineering, and operations to agree on the precise scope and outcome of a project. It also requires the operations side to address issues such as balancing asset utilization with flexibility. For example, high flexibility can be achieved by either having multiple pieces of diagnostic equipment in a hospital, each doing a specific analysis, or a few multipurpose machines capable of performing many procedures. The former case carries higher costs while the later may result in a long line of customers/patients waiting for its use. Simulation provides the platform to effectively address these types of issues.

Detail design

Simulation provides tangible benefits during the detailed design of any proposed new operation or change to an existing operation. Savings of 3 to 30% are typical. For example, a major equipment manufacturing company saw a 15% reduction in engineering costs as a result of simulations providing a faster resolution to design issues. Savings also accrued from simulation's ability to provide consistent, reusable engineering design and documentation. Since this company guaranteed performance of their equipment, the simulations meant faster testing and startup periods.

Less tangible, but still quantifiable, benefits are savings from design errors that are caught early in the final design phase. A soft drink manufacturer decided to add another bottling line. Cases from the line would be sent to a palletizer that six other lines were already sharing. Based on historical data, the palletizer would have the capacity to handle the new line. However, a simulation taking line dynamics into account showed that the single palletizer would actually be a bottleneck and reduce the plant's total output. A net savings of nearly \$600,000 was attributed to finding and rectifying the problem. Savings were based on estimates of the cost of modifying the packing lines after the incorrect design was completed, loss of production time due to the modifications, and loss of profit from being bottlenecked by the palletizer. The net savings were those costs minus the cost of the additional palletizer.

- Faster resolution of design issues or changes
- Consistent, reusable engineering design and documentation
- Reduced cost caused by design errors
- Faster testing and startup

Figure 1.9 Design/Execution savings (3-30% of project costs)

Making the correct design decisions early and quickly improves project efficiency and cost effectiveness

Simulating operations

Benefits from simulation of actual operations typically return savings of 5 to 25% based on reduced operating costs or increased throughput. Much of the savings are predicated on a simulation being able to reduce analysis time, thus providing a quick method for trying out changes or new ideas. Such analysis is especially important when shared equipment and staffing issues are involved.

Most importantly, a simulation can show the impact of an operating schedule on performance and guide management in obtaining optimal operations. For example, simulation showed a consumer products company that by simply changing the transfer logic between their batch operations and filling line, they could gain a needed 15% increase in production without any capital outlay. Similarly, a shipbuilder gained 10-20% in throughput in a bottleneck shop by re-sequencing work.

- Reduced analysis time
- Dynamic representation of operations
- Simple and quick method for trying changes or new ideas
- Accurate assessment of schedule impact
- Quickly analyzing the impact of shared equipment and staffing issues

Figure 1.10 Operations (5-25% of project cost)

Being able to study the dynamics of system operations leads to better understanding and improvement

Section 1-5 Simulation users

Not everyone who uses simulation will have the time or inclination to become proficient in all of its aspects. As discussed in this book, there are many requirements for a simulation project to be successful; as a result, a wide variety of skills are needed. However, in order to contribute to, and benefit from, a simulation project everyone does not need to know all aspects. Users should be able to find parts of a simulation project that match their needs as well as their abilities. Users should also be able to increase their simulation knowledge at their own pace. To facilitate this concept, the remainder of this book follows the progression of a user from having a casual relationship with simulation to one who is deeply involved in advanced topics. The benefits, however, are available to all levels of users.

Occasional User

The Occasional User is a person who only uses simulation to answer specific issues or to occasionally analyze an operation. At this level, simulation is oftentimes only used once or twice a year. Training must be short, the interfaces intuitive, and re-training (when simulation is again used after an extended lapse of time) minimal. Occasional Users are often managers, business unit leaders, marketing personnel, engineers whose main responsibility is not simulation, or operations personnel such as team leaders. Usually these users have a run-time or limited-use license for the

The most basic user may use a simulation infrequently and then for only a specific purpose

simulation application. They are most likely the “owner” of the problem or issue to be analyzed by simulation. They typically have much more knowledge of the purpose and domain of the simulation than of the simulation application itself.

Occasional Users can open and run pre-built simulations. The user interfaces are usually custom built to make controls intuitive to the user. By running the simulation, the Occasional User can validate the simulation against historical data or check the operating characteristics of the simulation. They provide machine data and other operating information for various scenarios. Their interaction takes place through custom interfaces, data tables, and separate applications, such as *MS Excel*. Details such as batch recipes and production schedules can also be manipulated in a similar manner. Analysis reports are usually custom built for the Occasional User, although the user interface may allow changes to the report structure.

To get the most out of a simulation, the Occasional User may define or help specify a simulation that a more Advanced User can build. For that purpose the Occasional User constructs what is known as an object flow diagram (OFD) which defines material flow, operating parameters, and logic. A value stream map may act as a starting point, but the OFD is required to add information that is not included in the standard value stream map. The method for building an OFD is detailed in a later chapter.

The skills the Occasional User needs begin with a basic understanding of simulation and its use. This user must also understand the nature of building a simulation and steps and people involved with the process. The Occasional User should also be familiar with modifying data in tables and *MS Excel* for both input and report functions. The Occasional User must have sufficient understanding of the system being simulated to make rational decisions about changes to the input variables and well as understanding the output and performance measures.

The capabilities of an Occasional User and the required skills are summarized below.

Occasional User capabilities

- Open and run pre-built models
 - Validate system operating characteristics by observing the simulation as it runs
 - Change data for objects and operational scenarios
 - Use pre-built reports to analyze performance measures
- Help construct a visual Object Flow Diagram (OFD)
 - Document that operating characteristics are correct
 - Participate in a review of the OFD
- Be an active participant in the various roles associated with the simulation project

Occasional User skills

- Have a basic understanding of simulation
- Open pre-built simulations
 - Use interface menus
 - Change object and operating scenario values
 - Analyze simulation output and object statistics
- Understand the development of an OFD
- Understand the steps in a simulation project

Intermediate User

The Intermediate User is an active simulation practitioner and typically has some problem solving role as his or her primary focus, such as an engineer in a corporate or operations group. Intermediate Users are also found in logistics, research, and business unit functions. For this user, simulation is a tool that is available to help with the person's main responsibility.

The Intermediate User has all the skills of the Occasional User and works closely with that person to analyze systems and resolve issues. Like the Occasional User, the Intermediate User can work with existing, pre-built models but in addition is able to expand or modify them. This user can take a OFD prepared by the Occasional User and add connections and basic logic to equipment in the model utilizing pre-built programming templates. The logic includes setting the sequence of events for operations and the logic that determines material flow paths.

The models that the Intermediate User builds include transportation details such as operators, automated vehicles, fork trucks, etc. This user can also modify the visual characteristics of the simulation. Functions of the model include describing the physical contours of conveyors, changing the physical characteristics of all objects used in the simulation (size, color), and controlling the way the simulation screen appears to others.

At this level of expertise, the user can open each object and modify its internal data based on knowledge of both discrete- and fluid-based processes. For fluid operations, the user can set up batch and blending recipes. The user also builds data collection reports based on basic templates and can export data to *MS Excel* or other programs.

While this level of user still doesn't require a programming background, the person must be willing to understand the basic software application structure and architecture. This structure may include concepts such as a software hierarchy, a tree structure similar to the standard *Windows Explorer* format. Other concepts include material and information flow connections between objects, the functionality of *triggers* that are executed when certain events occur, and the use of right and left mouse buttons. The user can also set up line and system controls that allow a simulation to run schedules and control when equipment is used. Finally, the Intermediate User has an

Most technical personnel will be able to use simulation as they need it and will take advantage of "ease-of-use" attributes of simulation software

understanding of the tools used to set up and analyze experiments and define performance measures in order to assess various alternatives and operating scenarios.

The required skills and capabilities of an Intermediate User are listed below:

Intermediate User capabilities

- All capabilities and skills of the Occasional User
- Assist the Occasional User to use simulation and analyze system
- Build basic models
 - Discrete models: including operators and transportation
 - Fluid models: including batch and blending recipes
- Expand/modify existing models
- Add basic logic and material flow information using programming templates
- Modify the visual appearance of models
 - Size, color, and shape of objects
 - Conveyor configuration
 - Specific simulation views

Intermediate User skills

- Understand all object functionality
 - Menus
 - Triggers
 - Interobject connections, e.g. input, output, center ports
- Build data collection reports from standard templates
- Establish data tables
- Exchange data with *Excel*
- Set up and analyze experiments
- Apply logic using standard templates

Advanced User

Building on the skills of both the Occasional and Intermediate User, the Advanced User can build complex models with extensive custom logic. While some organizations may devote a person to the simulation task, the primary responsibility isn't usually simulation alone, but some internal consultant function. This user will be called on to help simulate and analyze conditions or systems that are beyond the

scope of other users. This person is most likely the primary technical contact with the simulation provider.

The Advanced User may have an in-depth knowledge of some associated areas, such as reliability and the interaction of competing down times, data mining, statistical analysis, etc. In larger organizations, the Advanced User may be responsible for maintaining copies of the simulation software and for developing custom libraries. These libraries hold reusable simulation objects that reflect specific pieces of equipment or operations. This user can modify not only the basic characteristics of three dimensional simulation objects but can also modify and change the image itself. As a teacher/consultant, the Advanced User works with other users to continually develop simulation skills.

By knowing the underlying software architecture and having proficiency in computer programming, the Advanced User can add additional logic to objects, such as photo eyes on conveyors and complex maneuvers of transportation vehicles. This becomes an important skill when totally new logic, logic not covered by standard choices in the simulation software, is recommended for an operation. Most significant improvement comes when utilizing logic and concepts outside the range of normal thought processes. This level of user can also create custom reports, define performance measures, and develop custom control interfaces and simulations so they can be used by Occasional or Intermediate Users.

The skills of the Advanced User go beyond simulation to specific application knowledge. The type of specific application knowledge required includes reliability theory and application, lean manufacturing, and value stream mapping. Custom and advanced simulation functions are achieved through an understanding of the software's programming structure, the use of data tags, local and global tables and variables, and internal messaging techniques. The ability to create new objects may also require skill in using three-dimensional drawing tools.

The capabilities and skills of an Advanced User are listed below:

Advanced User capabilities

- All capabilities and skills of Occasional and Intermediate Users
- Build complex models
 - Custom logic for equipment sequence of operation
 - Custom rules for material transfer
 - Advanced line and system controller use
- Build custom reports
- Add functionality for reliability and statistical analysis
- Assist the Occasional and Intermediate Users in building and analyzing simulations
- Build new objects for inclusion in custom user libraries

Larger organizations may have people well versed in simulation software. Smaller organizations may outsource their advanced simulation needs

- Adapt models for use by Occasional and Intermediate Users
- Change three dimensional representation of objects

Advanced User skills

- Advanced application functions
 - Reliability theory and application
 - Statistical analysis
 - Lean and value stream mapping techniques
- In-depth use of a hierarchical tree structure
- Set up line and system controllers
- Modify existing reports
- Batch and blending recipe structure
- Programming skills
 - Scripting language
 - Use of labels (attributes)
 - Use of messaging
 - Application file structure
 - In-depth use of the object tree structure
 - Troubleshooting techniques
- Use of three dimensional drawing applications
- Maintenance of simulation application
 - Use of local libraries
 - Updating software as needed
- Expanded use simulation application tools
- Develop custom travel paths for transportation vehicles

The amount of people that a simulation-using organization employs to have a successful simulation project may range anywhere from a single user to a group of users with varying levels of simulation skills. Large organizations may have a distribution of users like the one shown in Figure 1.11.

Ideally, the vast majority of users would have both Occasional and Intermediate User skills. Modern simulation software makes it easier to obtain that level of skill. Since all skill levels are often needed at some point to obtain the most benefit from simulation, organizations with only Occasional Users will often outsource the advanced skills to the software vendor or third party consultants or engineering firms.

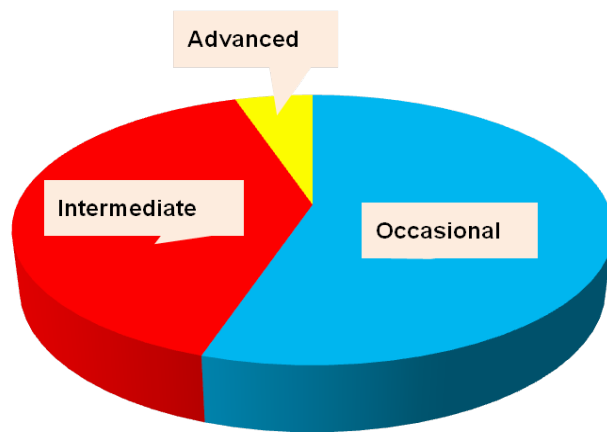


Figure 1.11 Typical distribution of simulation users

However, the value of simulation will only be achieved if an organization maintains at least one person at, or above, the level of Occasional User.

References

1. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, Chichester, England: John Wiley & Sons, Ltd.
2. Checkland, P. (1981) *Systems Thinking, Systems Practice*, Chichester, England: John Wiley & Sons, Ltd.

Chapter 1 Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Define operations system and provide some examples.
3. Identify some behavioral characteristics of operations systems that make them difficult to analyze.
4. Define simulation in terms of decision support.
5. Describe how model-based decision support systems enhance the decision-making process.
6. Describe the paradigm shift that has occurred for how simulation is used.
7. Identify some ways in which simulation is used in everyday life.
8. Describe some areas besides manufacturing that use simulation.
9. Describe the two basic tenets of simulation.

10. Provide examples of benefits that result from using simulation; consider each of the four benefit categories.
11. Describe the skills that are desirable in each of the three types of simulation users.
12. Describe the capabilities commonly found in each of the three types of simulation users.

Chapter

2

Simulation Applications

Before learning how to build simulation models and analyze their results, it is important to understand what models are and how they can be used. This chapter investigates the many types of operational problems that simulation can, and has addressed. The application examples are certainly not exhaustive, but should serve to demonstrate the versatility and power of simulation. The chapter also provides a brief summary of the simulation software marketplace in order to put *FlexSim*, the software used in this book, in context. The chapter also discusses the role of simulation in conjunction with other types of applications found in most enterprises.

Section 2-1 Modeling and simulation

The role of simulation has been known for some time; however, it's only been since the early 1980s that simulation applications have reached a point where they are truly accessible to a wide range of users.

Modeling as a starting point

Simulation starts with a model. A model in this sense is defined as

a physical or mathematical description of an object or event.

A model usually represents a single point or action in time. A simulation then puts the model in motion. That is, they are dynamic—their states change over time. An operating scale model of a piece of equipment is called a physical simulation. Modern simulation applications are based on mathematical and logical models, but can look physical as well through the use of three-dimensional graphics.

Static modeling has its origin in ancient time.

Some of the earliest examples of models were representations of physical buildings created to convince a pharaoh or king to approve a project. Time and random events had no meaning to those models.

Artist renderings of buildings today are visual models at a point in time. Until the early 1970s, static models of large oil refineries were created before construction began. These scale models were created using miniature pipes, tanks, and other equipment built to scale. Such models were used to provide a three-dimensional view of a plant and check the proper placement of the equipment and the hundreds of miles of pipes criss-crossing the plant.

1. Static Models
 - Clay figures
 - Artist rendering
2. Working Physical Models
 - Miniatures, prototypes
 - Pilot operations
3. Mathematical Models
 - Steady-state stoichiometry
 - Queuing theory and stochastic processes
 - Neural networks
 - Expert systems
4. Simulations
 - Continuous
 - Discrete event

Figure 2.1 Static models to simulation

Mathematical models focused on a basic material balance and are used extensively in the chemical, oil, and gas industries.

The industrial revolution brought with it physical simulators. Manufacturing companies hired master model makers who could build a working representation of a piece of equipment. The equivalent physical simulation for chemical production was a small sized pilot plant. Like the machine models, the small physical representations had an element of time and dynamics associated with them but were unable to capture the random events surrounding the operations and were difficult to scale-up to actual size. Both the working model and the small representations became expensive to create, and their value was diminished by their limitations. Mathematics replaced the physical representations.

Mathematical models come in many forms and are usually associated with the systems they are trying to describe.

The first form of the mathematical model was used in fluid production facilities, such as chemical plants and refineries, and came in the form of equations defining the physics of material transfer. Undergraduate Chemical Engineering departments teach the basics of stoichiometry starting with the material balance equation

$$\text{input} = \text{output} - \text{change in accumulation.}$$

Processes were considered to be operating in one of two modes: steady state or transient. At steady state, variables do not change, and the accumulation term falls to zero; thus, the material balance equation becomes

$$\text{input} = \text{output.}$$

This level of equation, or model, was used extensively for design calculations and analysis—especially in the petroleum and chemical industries where plants would produce the same product for long periods of time. Many companies still

Many methods have been used to extend mathematical modeling to better represent dynamic systems.

use versions of the material balance equation in spreadsheet form to model operations and calculate performance metrics. Today it is recognized that the steady-state assumption is not valid over time as operations are constantly subject to changes in variables and interactions. The steady-state calculation, although useful in gaining insight based on averages, is no longer sufficient to answer questions based on dynamics and statistical variations.

There was also a need for dynamic modeling in many types of manufacturing. Fabrication assembly industries work in a continuous manner, but their activities can be broken up into individual events.

Discrete event systems and queuing theory

The term *discrete event* refers to

those operations that have events occurring at distinct points in time and changing a state of a system.

Such events could consist of setting up for a machining operation, requesting an operator for assistance, the entering of a piece of material into a machine, and the leaving of a piece from a machine. An example of some discrete events in a banking operation may be a customer entering the bank, choosing a teller line, getting to the teller, requesting a certain type of service, leaving the teller, and leaving the bank.

Industrial engineering curricula include various ways to mathematically describe the actions of discrete-event systems. For example, through queuing theory, random processes are described in terms of assumed probability distributions. As a result, performance measures—server utilization, mean number and time in the queue, and mean time in the system—are calculated directly.

The basic components of a simple discrete-event queuing system are represented in Figure 2.2. In this case, the system is driven by the random events of a customer arriving and completing services. In fact, the basic parameters of queuing models are arrival rate and service rate—both of which are typically random variables. Examples of system logic include how the customers queue while waiting for service and how, in terms of the order, the server selects customers for processing. Such methods capture the long-term impact of variability and provide estimates of the performance of these dynamic systems; however, their limiting assumptions make them considerably less representative of more complex systems.

Expert systems are another example of computer-based mathematical modeling. They are based on cause/effect and rule-based relationships. Expert systems characterize and model operations based on large systems composed of rules and logic statements, typically in the form of if/then scenarios. In this approach, personal experience is captured from individual “experts” or historical data. In any case someone has to identify the cause and the related effect.

Manufacturing operations, which can be defined in terms of discrete events, utilize queuing theory for analysis.

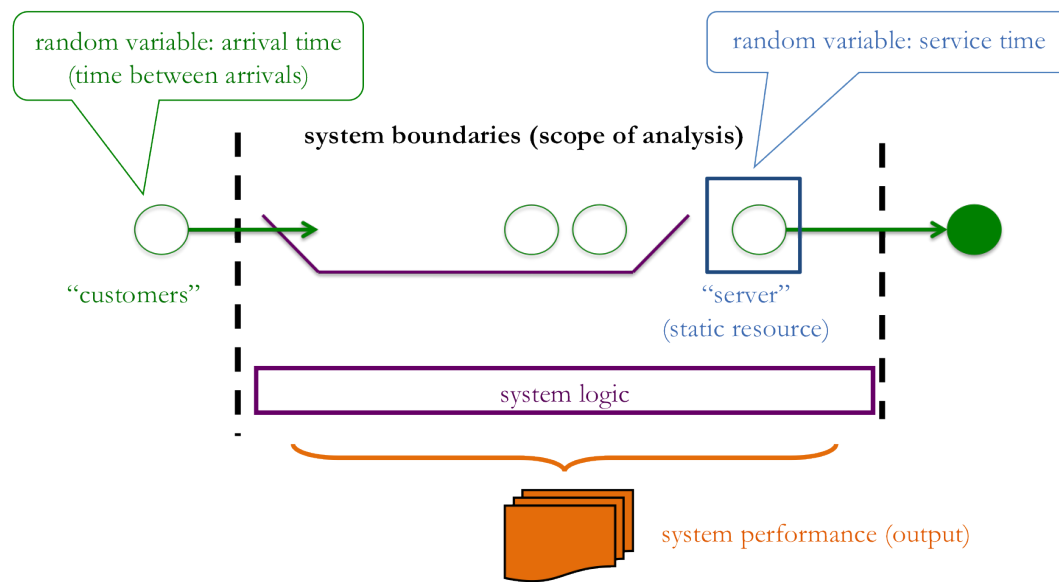


Figure 2.2 Representation of a basic queueing system

When only data are available and the underlying structure and relationships are not known, statistical regression analysis, fuzzy logic, and neural networks can be used to identify and define mathematical relationships. These methods require extensive quantities of valid data. While valuable in many situations, these methods tend to lack the robustness needed to handle complex interacting dynamic systems. Also, they indicate correlations and mathematical “fit” between the data and hypothesized functions, but do not necessarily capture causality.

Although models in general—whether regression, queuing theory, simulation, etc.—are widely accepted, two rather paradoxical comments point out the strength and weakness of trying to describe actual processes with models:

Models are tools to support and extend the power of thinking.¹

Essentially, all models are wrong, but some are useful.²

As this paradox points out, models can be great thinking tools, they foster discussion and understanding of how systems operate and encourage exploring and generating new ideas for improvement. Models are not intended to replace thinking, to the contrary, they are used to enable and enhance thinking processes. Of course, as the paradox also points out, since models are simply representations or abstractions of real systems, they do not, and cannot, capture all of the behaviors and complexities of real systems. However, replicating every detail of a system is not necessary for it to be very useful. In fact, trying to capture too much system detail in a model can make it unwieldy and complex. A model heavy in system detail becomes difficult to understand and maintain and thus is not useful. One of the arts of model building is deciding on the right level of detail to effectively deal with the problem being addressed. Simulations start with models then use and expand them to describe operations systems with complex interactions over a continuum of time. As an analogy, a model is to a picture, as a simulation is to a movie.

For the type of modeling referred to throughout this book, we adopt Pidd's definition that a model is

an external and explicit representation of part of reality as seen by the people who wish to use the model to understand, to change, to manage and to control that part of reality.¹

Section 2-2 When to use simulation

Not all operations have to be simulated. The objective of a simulation is to analyze operations and obtain results. The results are most often in the form of metrics. These metrics are key variables or calculations, based on the system's behavior over time, that are needed to resolve issues or indicate levels of performance. If the operations are in a simple sequential order, without unexpected downtimes or special events, metrics can simply be calculated using a spreadsheet; however, there are several operating characteristics that should raise a red flag and indicate that the calculation of metrics is not obvious and may vary widely due to statistical variations or other events.

Every system doesn't have to be simulated. Knowing when to use simulation is important.

The Degree of complexity

The main determining factor for deciding whether or not to simulate is the degree of operational complexity. Even the most lean and simplified operations may be considered complex if they are influenced by external decisions, such as production schedule changes; random events, such as breakdowns; or their own flexibility that allows them to be changed frequently.

Oftentimes manufacturing and other operations depend on the characteristics of the product or service being provided. This can add operational complexity, especially if the arrival sequence to the system is random, or at least appears to be random, and has a high degree of variability. For instance, a machine may be able to produce a variety of parts, but its setup and processing times depend on the type of part being produced. Of course, the processing can vary from part to part within a type. Another example is a shopping checkout line where the time to tally the goods depends on how many people there are and if additional steps (e.g., weighing vegetables or checking coupons) need to be performed. The method of payment (e.g., check, cash, credit card, food stamps) also induces variability.

Even a lean manufacturing design can prove to have complex dynamics if impacted by random events. Calculating the performance of a simplified processing line is more difficult if it is subject to part shortages at random times, or if it is dependent on operating personnel to clean the equipment periodically before subsequent steps can be completed—especially if the personnel are shared and have other competing tasks to do. Lean operations that have any degree of batch processing may use simulation to help optimize the batch size under various conditions.

Application Examples

Bottlenecks can change dynamically, and simulation is a good tool for analysis.

The following “Baker’s Dozen” of examples, based on actual experience, show how simulations are used to address various operational issues. Each issue includes a screenshot of a simulation model used to address the issue.

Issue 1: Understanding the bottlenecks that limit an operations system’s performance.

Every system has a bottleneck that restricts how well the system operates. Some companies base their performance calculations on the operation downstream of the bottleneck. Bottlenecks in processing operations are often identified and known; however, they are often transient and change as the system dynamics change.

There are also many ways to alleviate a bottleneck. In manufacturing, equipment from another production line can be used. In a supermarket, a new checkout line

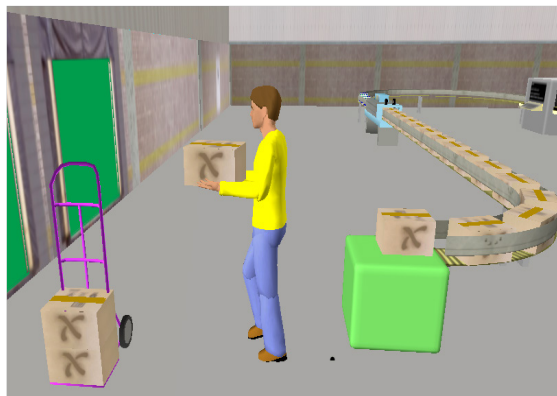


Figure 2.3 Bottlenecks can appear anywhere

can be opened. At times there is no single bottleneck to focus on. Whenever a step in a process stops, it becomes the bottleneck. For a restaurant, the bottlenecks may change according to the seasons. Dynamically changing bottlenecks makes a system very difficult, if not impossible, to analyze with simple spreadsheet methods. The dynamic focus of simulation makes it an excellent tool for bottleneck analysis.

Issue 2: Addressing the accumulation of material in a manufacturing process.

Modern manufacturing theory focuses on the elimination of any unnecessary steps or accumulation of work materials in a process. Surge areas, which are used to store

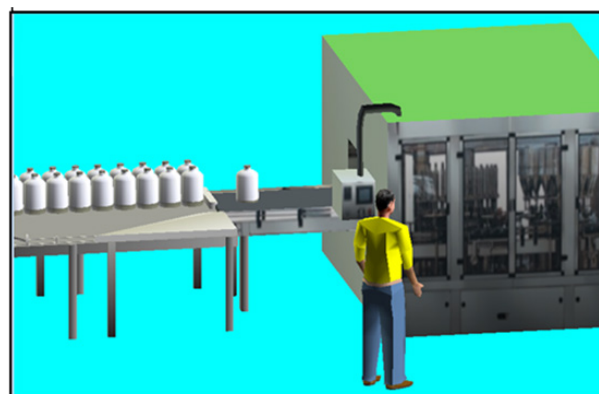


Figure 2.4 Accumulation table surge before a filler

materials ahead of equipment or workstations, are targets of Lean thinking; however, Lean operations do not necessarily mean the eliminations of all surge areas. Point of use storage is still an integral part of modern, efficient operations.

The sizing of surge areas and buffers are not intuitively obvious as they are based on the dynamic behavior of the operation. Simulation is used to

Elimination of all surges may be an ideal but can lead to trouble—simulation shows how to correctly manage the size of surges.

identify where the surges are needed and what size they should be so that a minimal amount of inventory is accumulated in a production area.

Issue 3: Analyzing new concepts.

Simulation is often used to analyze the impact of modified or new operating practices or policies. Testing new concepts with simulation is much faster and more economical than trying to do it on the actual system. The impact of reliability assumptions on equipment can be tested, as well as the new methods. If there are absolutely no historical data to use in the simulation, then the simulation can be used to conduct a sensitivity analysis in which variables are modified in order to study their impact on system performance. The results are used to specify the operational characteristics of the equipment or used to estimate changes in operation.



Figure 2.5 Evaluating alternatives before implementation

Testing new concepts on a simulation can not only be faster but far less expensive than building it and then trying it out.

Issue 4: Studying combined processing and packaging systems.

Systems that both process and package goods are prime candidates for simulation analysis. A major issue is that the process and packaging areas are closely coupled, making them highly dependent on each other. The complexity is increased further if the processing is fluid or batch in nature and the packaging line is discrete. A simple continuous blending system for paint that goes to a packaging operation is an example of a closely coupled system. Once the operation commences, disruptions at any point can cause significant production disruptions.

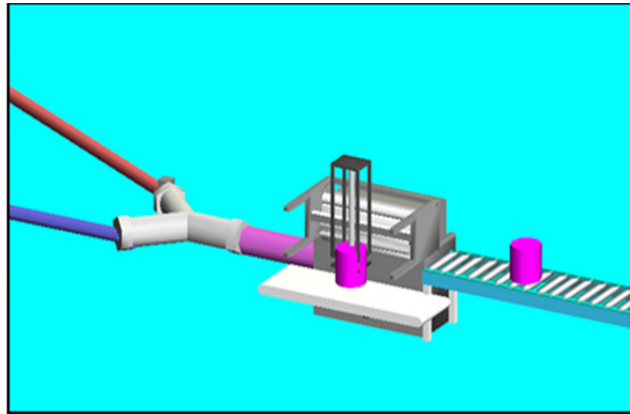


Figure 2.6 Paint blending and filling operations are closely coupled

Closely-coupled systems offer additional complexity that can be studied by simulation.

Issue 5: Balancing cost and service level.

At an airport security check point, travelers who have their tickets and identification verified may be sent to one of many inspection lines depending on several factors: ticket status, line length, and need for special screening. Simulation is usually used to design and analyze these systems. The simulations help determine the number of

Logic for flexible systems that have multiple paths for materials can be easily simulated



Figure 2.7 Improving airport security system operation

lines, amount of equipment, and staffing levels that should be used throughout the hours of operation. This determination must balance customer service with cost of operations. Simulation is used to evaluate the efficiency of such proposed concepts as frequent flyer lines and pre-screening background checks for certain types of travelers.

Issue 6: Validating complex material transfer procedures.

In an effort to make operations very flexible, material is often routed from one operation to another based on various rules. In a cheese plant, material is passed from fermentation units to processing areas based on the history of the receiving unit. These rules have to take into account the nature and age of the material in each location and the cleaning record of the receiving vessel. Simulation of such operating rules is used for training as well as ensuring that the rules do not become unintentional bottlenecks.

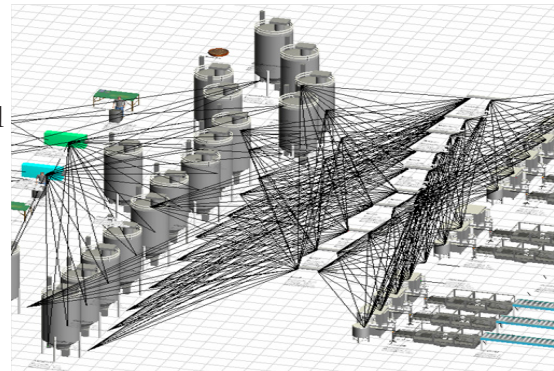


Figure 2.8 Multiple transfer paths

Issue 7: Checking batch operating sequences.

Batch operations contain a sequence of events that often change with the product. In fluid operations, the batch is usually described in a way prescribed by industry standards. The description starts with a recipe table listing the amount of each material to be added and a separate table lists the steps in the sequence. The use of a recipe and step table has become the standard for specifying batch operations.

For multiple products, separate recipe and step tables have to be maintained. Logic is employed to handle any additional cleaning required and to direct the output of the batch to its destination. Simulation is needed to analyze the number of batches that can be made especially if some of the steps can be subject to random events, such as the requirement of an operator or delays from test results.

Batch operations contain considerable logic and should be checked with a simulator.

Step Table			Recipe Table		
Step #	Description	Delay	Ingredient	Amount	Step#
1	Add Water	0	Water	2500	1
2	Add Citrate	10	Citrate	150	2
3	Mix	30	Premix	225	4
4	Add Premix	0	Water	1000	5
5	Add Water	65			
6	Mix	65			

Figure 2.9 Batch specification tables

Issue 8: Analyzing the impact of production schedules on operations.

Experience has shown that the greatest impact on resources and performance is the scheduling of products or people through the system. In manufacturing, a production facility is often at the mercy of a marketing or sales organization that dictates the order of material flow. Simulations that only focus on producing a single item or a set sequence of items are not robust enough to analyze performance over an extended period of time.

Additionally, when simulating a schedule, operations usually have to be adjusted for each item going through the system, and monitoring of the line is needed in order to know when the scheduled task has been completed.

Operations where more than one area that can be scheduled and the separate production areas can interact can be complex to analyze. Similarly, complexity occurs if the operations must be carried out in a particular sequence without exception and shared equipment is involved. In both these cases simulation provides a robust method that takes complexities into account.

At times the steps required to clean or set up an operation can be more complex than the operation itself. Food processing is an example of this as it has extensive requirements for sanitation and allergy cleaning that depend on the particular product as well as the elapsed time of operation. In addition to food processing, hospital requirements for sanitation and cleaning are also extensive. In all these cases, the time required to perform the activities varies statistically. Simulation takes these dynamics into account and allows performance metrics to be accurately calculated.

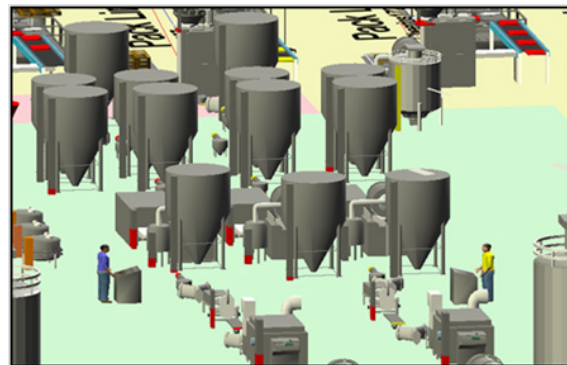


Figure 2.10 Two independent production lines sharing storage and packing equipment

Production scheduling is often taken for granted but may actually have a significant impact on performance.

Raw material receiving is often a neglected area but can be improved through simulation.

Modern production facilities are operated in a coordinated manner requiring extensive logic to optimize individual units.

Modern control systems contain complex logic which has to be validated and tested.

Issue 9: Improving raw material distribution within a facility.

The beginning or end of a system containing many operations is often a shared resource. A common example is the use of a shared raw material facility or shared warehouse area. While bottlenecks are known for the individual operations, the impact of the shared facilities is often neglected until it actually causes a loss in performance.

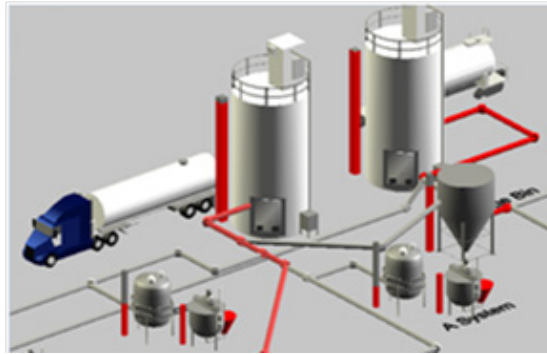


Figure 2.11 Distributing raw material to multiple plant sections

The speed of getting material from an incoming dock to its destination is often dependent on the routing and availability of transportation resources. The same is true of moving finished goods to their correct warehouse or loading dock destination. In medical simulations, shared resources of X-Ray or lab areas cause similar problems. Because of the interrelated nature of these operations, simulation is required.

Issue 10: Effectively coordinating a production line.

Coordination with other areas in a system is similar to the problems caused by shared facilities as discussed in Issue 9. In that case, however, each processing operation is still independent of the shared equipment. With coordination, two or more independent operations must interact at the exact same time. For example, in an automobile production line, components manufactured on independent lines must arrive at an assembly point at the same time in order to fill a particular order.

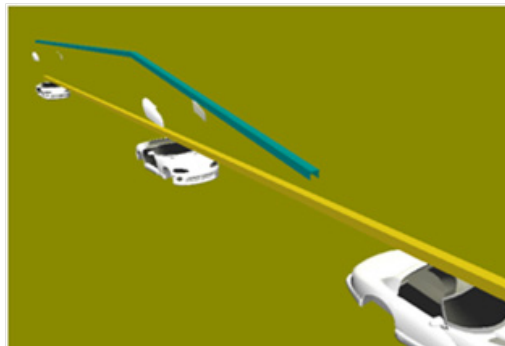


Figure 2.12 Automotive assembly lines require coordinated operations

In the food industry, various components must join to form “variety packs” of a finished product. The traditional way of dealing with such coordination is to use large surge systems. In a lean environment, this process is not considered efficient, and simulation is used to develop line configurations, operating procedures, and production rate strategies, as an alternative to surge systems.

Coordination of production areas is further complicated by the stopping and starting of parts of the production lines. Equipment downtime is generally considered to be the result of some failure; however, predicting downtimes is complex as there are often competing failures in different sections of a machine. Downtimes can also be caused by outside influences, such as operator break times, scheduled meeting times, and even preventive maintenance. Taking competing

downtimes into account and correctly entering them into a simulation is important for representing system behavior and conducting a comprehensive analysis. Simulation is a valuable tool for studying the impact of downtimes and how to overcome them.

Issue 11: Working with operations that involve people.

The behavior of people taking part in a simulated system is the most difficult aspect to characterize or model. Individual performance rates and logical choices vary widely. Simulation can take many of these characteristics into account when operators, fork truck drivers, bank tellers, health care personnel, ticket takers, airport security, or other people are involved. The improvement of health care facilities is of particular importance today. Simulation provides a method to analyze how patient needs can be both effectively and efficiently handled in facilities such as hospitals, emergency rooms and clinics.

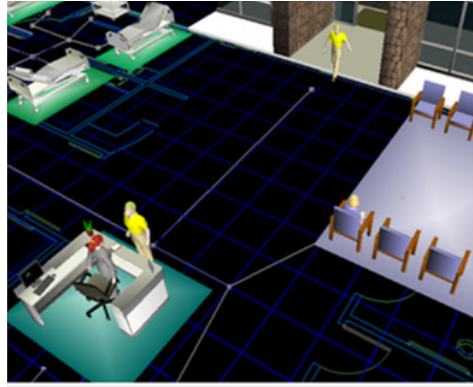


Figure 2.13 Healthcare simulations track people and their medical needs

Simulations can take the activities of people into account and focus on their service level.

Issue 12: Developing logic for material transfer.

Making significant improvements to operations can involve specialized logic normally applied through control systems. Such logic includes dynamically changing rates that depend on operating conditions, manipulating flows to maintain coordinated operations, or using photo eyes to control movement of material. Such logic can be complex, and accurately predicting its impact can be difficult without simulating the system.

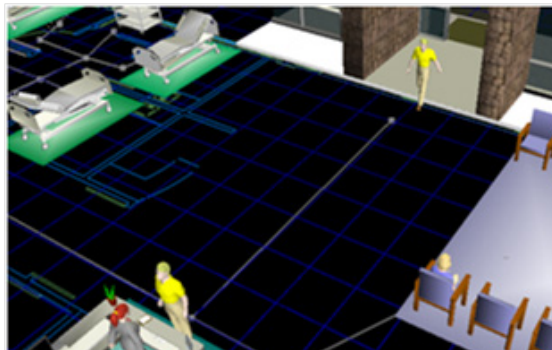


Figure 2.14 Conveyor photoeyes and logic control material traffic

Issue 13: Designing for just-in-time manufacturing.

Providing goods and services for a just-in-time production system means producing materials at a rate that matches the market demand. Such an operation is also called a “pull” system and requires precise control of the manufacturing process. Whenever demand changes, the processing systems must adapt to the demand in both quantity

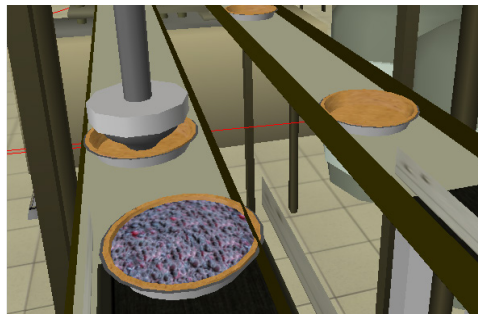


Figure 2.15 Pies are made to order

and quality. Significant logic may be employed to assure that the desired product is correctly managed. Additional logic may require that one operation be suspended to produce another product and then return to the prior operation. A pull manufacturing environment is best designed and analyzed with simulation.

Degree of complexity for various operations

The previous examples indicate complexity in operations and should raise red flags signaling that simulation should be used for design, analysis, or validation. Figure 2.16 summarizes a number of conditions which can increase the degree of operational complexity. Simulation is especially useful when the degree of complexity is moderate or greater.

The degree of complexity involved with a system is a good indicator of the need to simulate.

System characteristic	Degree of complexity
Straight through operations--Known rates--known upsets	slight
Variables dependent on service, product, or operating conditions	slight-moderate
Lean design with random events or external changes	moderate
Bottlenecks with by-pass or alternative logic	moderate
Bottlenecks that change by conditions	moderate
Surge or Kanban areas needed	moderate
Introduction of new operations or services	moderate
Reliability important--failure modes, interactions	moderate
Interaction between fluid and discrete manufacturing areas	moderate-high
Multiple possible material transitions or transit paths	moderate-high
Batching operations	moderate-high
Scheduling influence on performance	moderate-high
products or services must be executed in a particular sequence	moderate-high
Specialized setup or cleaning requirement	moderate-high
Shared equipment or services	moderate-high
Coordination required with other areas or people	high
Competing downtimes (e.g., reliability, break or lunch times, shut downs)	high
Interaction of services, equipment, and people	high
Specialized logic (e.g., dynamically adjusting variables based on condition)	high
Pull manufacturing (make to order) operations	high

Figure 2.16 Degree of complexity of typical operations

Section 2-3 The simulation software marketplace

Early commercial simulation programs focused on specific areas, such as discrete event simulations or fluid/continuous simulations. Also, because of the computer and software technology available at the time, they were not user-friendly and required a long learning curve. Consequently, groups of simulation specialists were formed to develop simulation models and analyze the results. Many companies had large groups of simulation practitioners who focused on detailed simulations. When more individuals within the organization wanted to make use of simulation technology, they were disillusioned by the learning curve of the applications; the license costs; the time to complete a simulation; and the placement of a separate group between them, the process, and the problem they wanted to study.

Commercial simulation applications initially required larger computers and extensive specialized training.

Simulation languages

As early as the 1960s and 1970s, simulation languages and simulators were being developed. High-level languages such as *SIMULA*, *GPSS*, *GERT*, and *SIMSCRIPT* started to replace the lower-level language of *FORTRAN* as the language of choice for simulation. These simulation languages were designed to have constructs that made it easier to develop simulation models.

When developing simulation programs using these early simulation languages, the modeler had to use language statements, or macros, to specify the system. Data were passed to these constructs through a set of arguments, and the modeler had to understand what each parameter in the arguments list meant and keep track of the system state variables. The language constructs or macros were not very robust and oftentimes required writing supporting code in *FORTRAN*. Simulating systems that required anything more than the basic functions could take a long time and be very expensive to develop.

Early models that utilized simulation languages did not have animation. The first language to add animation was *SIMAN*. *SIMAN* had to use a separate second language called *CINEMA* to build animations. *SIMAN* and *CINEMA* were later combined and called *Arena*.

The first simulation software products for DOS-based PCs were primarily for commercial use and were not widely used or even available until the early 1990s. By today's standards, simulation products during these early days were hard to use, had very limited graphic capability, and pushed the limits of the computers. 3D animation was virtually unknown and required very expensive workstation computers that could cost up to \$60,000. *AutoMod* and *QUEST* are examples of early simulation packages that could support a level of 3D animation. The cost of hardware and software to run the system could approach \$100,000. Even then, the model ran slowly and required extensive programming.

In 1986, *WITNESS* released the first product with higher level prebuilt constructs that also had a visual representation. These advanced constructs further reduced

The advent of the PC and advances in programming technology brought simulation applications within reach of a broader audience.

Current simulation applications available on the commercial market utilize object-oriented programming methods and 3D graphics engines.

the need for direct programming in simulation languages. The new generation of simulators were easier to use, but still lacked flexibility. Modifications to the default constructs were either very difficult or impossible to accomplish because the computer code behind the construct was not available to the user. By the late 1980s, there were several PC-based simulators on the market, including *WITNESS*, *ProModel PC*, *SLAM*, and *SIMFACTORY*.

Simulators

In the 1990s several newer simulators began to appear in the market. In Europe there was *Simple++*, *Simul8*, and *Taylor II* (the first 3D simulator to run on a PC). Several of these simulator products are still being used today as a result of the availability of textbooks and student versions.

In the United States, where the majority of simulation software was used commercially, several small companies were introducing a variety of products. Some products were short lived, and others are still being used today. The more notable ones are *Extend* and *Simcad*. In 1998 Taylor ED was released. It was the first object oriented 3D simulation software that had virtual reality 3D. It also operated on the Microsoft Windows software platform.

FlexSim was released in January 2003, and proved to be substantially different from previous simulators. It was also a departure from the standard simulation language paradigm both in its usage as well as its underlying architecture.

While these simulation languages and simulators were primarily focused on discrete-event simulations, applications for the fluid/process industries, such as chemical plants and oil refineries, started to appear in the market.

For the fluid industries, the fundamental principles of dynamic material and energy balances and computational fluid dynamics were used to create time-variant simulations of very specific operations. These applications are used to simulate such conditions as temperature profiles in jet engines, moisture migration in dryers, and pressure profiles in extruders. *Aspen*TM became the premier simulator for the chemical and refining industry. That application creates detailed simulations of transformational events associated with chemical processing.

Hybrid Simulation

Not all simulated systems fall exclusively into either the discrete event or fluid categories. For example, many fluid operations contain a batching or blending step where the products' components are added to and leave the equipment as fluids; however, they are then placed into discrete units such as bottles or boxes. Operations that combined both approaches are termed "hybrid" systems.

Some simulators have the capability to handle both discrete and fluid events in the same application. *FlexSim*, *Extend*, *Arena*, and *WITNESS* are examples of such appli-

cations. They all use different techniques to simulate continuous, fluid operations in a primarily discrete-event simulation environment. The main characteristic of a fluid operation is the specification of material processing in terms of the flow of material per time unit (e.g., gallons/minute) rather than as a discrete cycle time.

Additionally, *fluid* might not mean a liquid. In the production of potato chips, the flow of chips is characterized by a flow in chips/minute or pounds/minute. In high-speed packaging operations, fillers are rated at bottles/minute. Trying to simulate each individual piece of material in either of these situations would be very time consuming and typically would not add much value to the analysis.

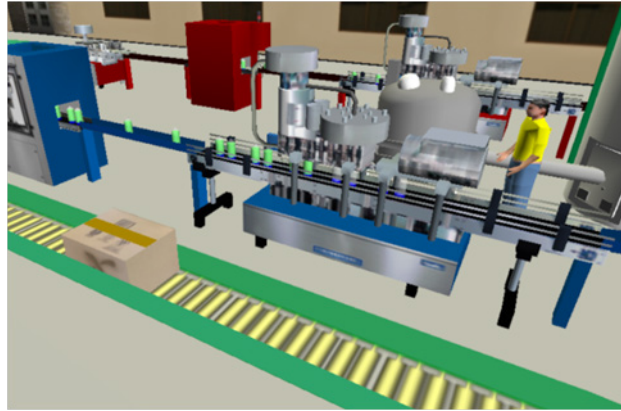


Figure 2.17 Bottling line is both fluid and discrete

A basic way to handle fluid flow in a discrete simulation is to “chunk” the material; for example, ten gallons of fluid could be represented by one discrete unit. While technically possible, changing chunk sizes dynamically can be difficult. This would be required when putting the fluid material into a bottle or bag, or moving it into and out of a tank at different rates.

Another approach is to calculate the accumulation of fluid material at intervals. The change in accumulation can be calculated by knowing the material flow rate and the time between calculations. These calculations are complicated if various pieces of fluid equipment are in series (e.g., tanks, pipes, processors). In these cases, calculations have to be recursive to be able to tell if flow is blocked or a pipe is drained.

Today there are two common approaches for calculating fluid properties in a discrete simulator. The first is to make a calculation whenever a discrete event happens in the simulation. This method is efficient but may not be accurate between discrete events. The second approach is to create events at specific intervals (e.g., every clock tick). This method is similar to using a scan frequency in process control systems, and while more accurate, it adds computing time to the simulation.

The Trend toward “Ease of Use”

Since the year 2000, the simulation marketplace has dramatically changed. Companies, seeking to reduce costs, eliminated their simulation groups with their high overhead costs. While still seeing simulation as a valuable technology, companies found it difficult to justify the overhead associated with maintaining a separate simulation organization. The combination of a new level of simulation software and more powerful, less expensive computers helped to create a change in the marketplace. The new key attribute that modern simulators strive to achieve is known as *ease of use*.

Looking for ways to reduce cost, companies found they could not justify large simulation groups.

This trend has helped simulation to become a more common tool that is no longer focused on simulation professionals in large corporate groups.

While *ease of use* is basically a marketing term, there are a few basic characteristics generally associated with the term; some of these are provided in Figure 2.18.

1. Intuitive operation
2. Consistent use of terminology
3. Short training and re-training periods
4. Allows the focus to be the application
5. Robust
 - Multiple user capabilities
 - Fluid and discrete modeling
 - Basic and advanced programming
6. Re-use
 - Models
 - Individual constructs

Figure 2.18 Ease of Use characteristics

Simulators that are considered easy to use incorporate modeling procedures that are intuitive and maintain consistent terminology and interfaces. Following a *Windows* approach to file structure helps to accomplish this, as do user interfaces that can be customized to reflect the user's level of simulation experience.

Visualization is an important factor. While not always required, a good three-dimensional representation

can aid understanding of the simulation by managers, operators, and others who are not involved in building the simulation. Visualization also makes it easier for the simulation builder to compare the simulation to actual operations.

Training is a major issue for software. Many users are familiar with basic computer applications and gaming systems. They expect to be able to use a program without reading the instructions. Simulation software certainly requires training, but it can be done at a variety of levels that are aligned with users' roles in the simulation project. Training should be short in order to get a large number of users started with a base, then that base should be easily built upon as users gain experience. For users who do not use simulation every day, refresher training is often needed. This level of training should also be minimal. Simulators that can be used with little training, free the user to focus on the system being simulated rather than on the mechanics of using the simulator.

Simulators in the current marketplace also try to be robust—meaning that while they are easy to use for the new user, they can also provide advanced features for the expert user. Robustness also means that the simulator can handle fluid, discrete, and hybrid simulations in a consistent manner.

Reusability is a key factor in keeping engineering costs low. Development work that is done for one application should not have to be duplicated on another. The ability to create and keep partial and full versions of simulation models is important. Of course, if the simulator is not easy to document, then past work may be lost because no one will be able to tell how it was built.

FlexSim was selected as the simulator for use in this book because of its robust architecture and ease of use. It was used by Dr. Malcolm Beaverstock, the first author, to establish simulation as a common tool for use within General Mills by individuals with various levels of simulation expertise. It has also been used by Dr. Allen Greenwood, the second author, and the simulation group at Mississippi State University to help industry address a wide range of problems through the use of simulation.

Section 2-4 Simulation and other tools

Simulation is only one of a number of tools available for effectively studying dynamic systems. Other types of tools used extend from directing day-to-day operations to planning changes to meet new business opportunities. It is difficult, if not impossible, to find one tool that easily satisfies the needs of all users across such a wide range of activities; however, all of these tools share in the objective to improve system performance as they also share in the data and metrics that drive operations. It is therefore critical that the tools are able to share information across applications as well as organizational boundaries. Figure 2.19 shows the interactions between the various organizations and the tools that support the operations.

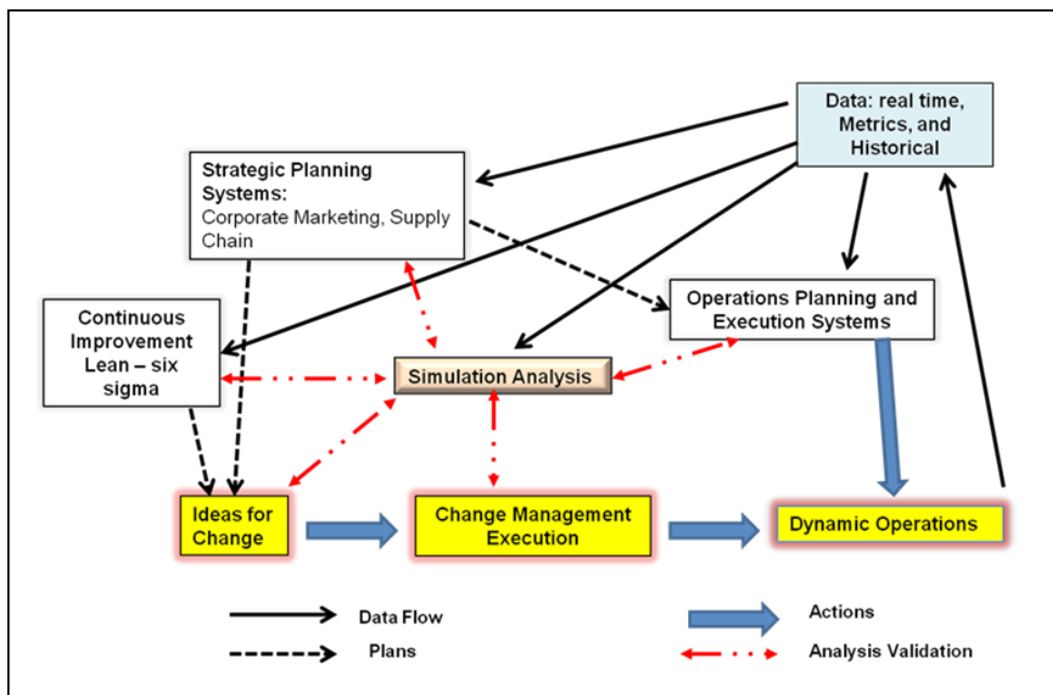


Figure 2.19 Interaction between simulation and other applications

The Dynamic Operations block can be a production line, a plant, a service organization, or even a single operation. It is this operation that is the focus of the improvement efforts. Understanding its dynamic behavior and performance measures is the underlying goal. Two forces directly change the operating characteristics of the system. The first force is the day-to-day directives for operation.

Each organization in a corporate environment usually has its own applications to help in carrying out their assigned functions.

These may come from a direct response to economic or market conditions or from events causing the system to react. The second force is the introduction of new products or services, as well as changes made in hopes of improvement.

There are many tools and methods available to direct the planning and execution of day-to-day operations. Major corporations use enterprise software applications such as those included in the *SAP*TM family of enterprise resource planning software. More focused applications such as *Supply Chain Guru*TM are also available. These applications monitor the current state of operations, inventories, and sales to determine what should be produced on a specific day at a particular location or manufacturing line.

Understanding the roles of various applications

In a small operation or service area, operational strategy can be as simple as having someone with experience in the operation using personal judgment to make decisions, or it can be as complex as requiring a software application to make decisions.

In either case, the decision method usually involves some form of simplified model of the operations. The model may be a mental model or a mathematical model and is used to transform business or demand inputs into operating decisions. Because models at this level tend to be generic in nature and do not incorporate the dynamics and interactions that exist, operating decisions will likely not be optimal for the enterprise and may actually increase the cost of producing the goods or services.

Another basic force that impacts an operations system is the physical or procedural changes to the system that are caused by the introduction of new methods, equipment, projects, or goods/services. While some of these changes can be

<ol style="list-style-type: none"> 1. Supports <ul style="list-style-type: none"> • Innovation <ul style="list-style-type: none"> • Lean, Continuous Improvement, Six Sigma programs • Project management • Prioritization 2. Validates <ul style="list-style-type: none"> • Design choices • Production schedules 3. Provides non-political sanity check for <ul style="list-style-type: none"> • Outside-of-the-box ideas • Research concepts • New ventures 	<p>executed with little effort by people closest to the operation, others involve significant resources and time. There may also be multiple changes being considered, thus requiring priorities to be set. Each such major change would require a justification for the cost of capital and possible disruption of operations. Most organizations have some form of change management process to oversee these projects.</p> <p>Project concepts come from a number of sources. The adoption of Six Sigma, Lean, and Continuous Improvement</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 2.20 Simulation and other activities

programs has resulted in people directly involved with operations across organiza-

tions recommending changes to improve performance. These changes are based on observations as well as analysis of real time and historical operating data. Tools developed for these programs are simple to use and do not involve operational dynamics or interactions. Change recommendations are typically local in nature. When local solutions are applied to systems that interact with others or have operational complexities, they may not be, and typically are not, optimal for the broader system, such as the enterprise.

Project concepts to modify operations that come from business or marketing groups tend to be expansive and costly in nature. These concepts are derived from model-based tools that reflect business, market, and logistic conditions. The driving force for justifications is heavily weighted toward business revenue. Operations may be modeled but oftentimes at a very basic level, usually only considering averages of rates and costs. Changes to operations are often major in scope and have to undergo various levels of justification. Their validation of operability (i.e., degree of confidence that the concept will actually work as planned) is usually minimal. These high-level concepts compete for resources proposed at lower levels in the organization that result from Continuous Improvement activities.

Simulation plays both a direct and supportive role in all decision-making activities that potentially impact the operation of a currently operating system. By using real time and historical operating data, a simulation analysis can generate ideas for change and improvement. Because the simulation characterizes the dynamic interactions within an operation, it is easier to visualize the impact of making changes. Simulations are most often created to seek answers to specific operating issues that seem to defy simple or conventional analysis techniques.

During the change management or project execution effort, simulation is used directly to analyze designs and estimate performance. A simulation can be used to determine the number of operating lines required or the optimal size of surge or Kanban areas. Simulations also check the control logic assumptions for dealing with various pieces of equipment, such as conveyors. The staffing levels needed for operations and equipment loading can be determined through the use of simulation. Performance of transfer systems for raw materials, warehouse layout design, batching control logic, and packing line analysis are all examples of simulation's direct involvement in supporting project design and analysis activities.

Simulation can also play a much broader role, in that it can be used to validate opinions or ideas. Simulation is used by companies with active Lean programs to both validate and help prioritize competing ideas that result from process improvement teams. Simulation also indicates where local optimizations may be affected by other areas and consequently will not provide the desired result. Lean teams view simulation as a quick and easy way to test concepts.

When simulation is used at an early stage with business and marketing groups, it can eliminate major projects that may not meet expectations. On the other hand, major projects are also often justified through the use of simulation. Outputs from simulation provide a more accurate basis for return on investment calculations since

Simulation has a unique role in working with other applications to help run an organization optimally.

it takes operations into account as well as interactions and variability. As a validation tool, simulation helps in the prioritization process where competing concepts and projects are involved.

Simulation is used to validate the results of execution and operations planning systems. Planning and scheduling tools that produce multiple week schedules do not necessarily take into account all of the operating factors. As a result, the schedule for an operation may meet some specified business goal but may increase the cost of operations. In most cases, business and operating organizations are independent and wish to maximize their profitability. By simulating the proposed schedule, operating costs can be more accurately estimated. By comparing schedule options with operating costs, a closer-to-optimal plan is possible since it addresses both business and operational requirements.

By interacting with all the various activities that impact operations, simulation plays the important role of a common communication and continuity tool. As Babin and Greenwood point out, simulation builds collective understanding of the process being improved.³ From idea concept to daily operation, simulation provides a means for people across an organization to visualize, analyze, and discuss operations and change concepts from a common basis. Discussions of priority or value can be compared on an equal basis. Rather than trying to discuss issues based on anecdotal experience, people can base their analysis on simulated results of the actual and proposed processes.

References

1. Pidd, M. (2003). *Tools for Thinking, Modeling in Management Science*, 2nd Edition, Chichester, England: John Wiley & Sons, Ltd.
2. Box, George E. P.; Norman R. Draper (1987). *Empirical Model-Building and Response Surfaces*. Wiley.
3. Babin, P, and Greenwood, A. “Discretely evaluating complex systems.” *Industrial Engineer*, February 2011, vol. 43, no. 2, pp. 34-38

Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Define model.
3. Describe the modeling paradox.
4. Select an operations system (i.e. one that involves queueing) and identify some events in that system that drive behavior and that would need to be considered in a simulation model.

5. Select an operations system (i.e. one that involves queueing) and identify some of the random variables in that system that would need to be considered in a simulation model.
6. Identify and describe some areas where simulation is typically applied to address operational issues.
7. Provide examples of highly complex operations that would benefit from simulation support in the design and analysis of that operation.
8. Describe some of the changes in simulation software and in the simulation marketplace that have occurred over the years.
9. Identify some of the characteristics of “ease of use” as it applies to simulation software.
10. Describe some of the ways simulation interacts with other applications.
11. Describe how simulation can be used in conjunction with lean and continuous improvement tools.

The Occasional User

Chapter

3

Using Simulation to Solve Problems

It is important for all stakeholders in a simulation to understand the power of simulation and how it can be used to solve problems. This chapter uses six diverse models to illustrate how simulation can be applied to improve system performance. All models employ a custom interface so that the users play the role of decision maker and Occasional User. The focus is on analysis and decision making and not on software and model building – that comes later in the book.

Section 3-1 Using a simulation

Simulation is an applied technology. There is little value to a simulation built without a reason or a problem to be solved; therefore, simulation must be experienced to be appreciated and understood. This chapter is an introduction to simulation applications through the eyes of an Occasional User.

An Occasional User, as defined by this book, is one who doesn't use simulation on a regular basis but appreciates the value of simulation. Usually this person's main responsibility is something other than simulation—a manager, lead engineer, team leader, etc. The Occasional User is not proficient in building simulation models but, with a little review, can make use of a simulation built by others, especially if the simulation is intuitive. It is assumed, however, that the user is familiar with some of the basic concepts of simulation.

The Occasional User knows enough about simulation to create a functional specification for others based on his or her knowledge of the system being simulated. As the “owner” of the problem or issue to be studied by the simulation, the Occasional User can detail the simulation requirements, define the scope, and establish the metrics for the analysis.

This chapter and the next will develop the skills and knowledge base for the Occasional User. Most simulators on the market today are set up for Occasional Users to run simulations and even provide “run-time” versions of the software. While *FlexSim* software is used in this book, the methods for doing the exercises in this section should be typical of other products as well.

The simulations used in the following examples represent models built by Intermediate and Advanced Users. They are usually prepared for Occasional Users to use and therefore can be opened and run without any knowledge of the underlying software application. Details of the application environment for *FlexSim* are contained in the Appendix.

Section 3-2 Simulations with random events

The challenge for the Occasional User is to understand the simulation he or she is viewing and to interpret the results. The simulation interface will normally allow the user to change certain variables and view previously defined metrics. For a simulation that contains random events, this can be confusing.

Almost every event in the world will happen differently every time it occurs. Each event is unique and influenced by its environment. This phenomenon is characterized in simulation models by using random events defined by probability distributions. For example, distributions are used to represent arrival processes, such as patients coming to an emergency room; process times, such as the time to assemble products; transportation times, such as the time to move material from one location to another; interruptions, such as machine breakdowns, etc. These statistical distributions can be developed based on historical data or by professional estimate.

As shown in Figure 3.1, and as described above, input to a simulation model involves random variables; therefore, the model is considered to be stochastic since it uses random variables. These stochastic inputs are a subset of the uncontrollable variables in the model; i.e., they are not under the control of the decision maker.

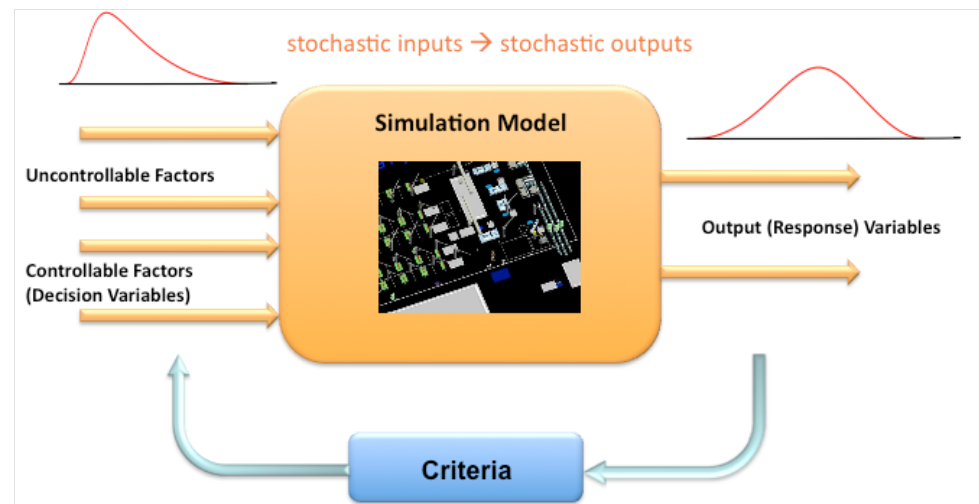


Figure 3.1 Simulation models are typically stochastic

Simulations
rarely, if ever,
are run only
once in actual
practice.

Controllable factors, or decision variables, are those that are manipulated in the model in order to improve a system's performance (e.g., number of workstations, number of transporters to move products, processing rules). Since the inputs to a model are stochastic, the output or performance measures are also stochastic. Care must be taken to evaluate stochastic output. One run of a model represents just one way a system may behave; therefore, a decision should not be made on whether the model is valid or if one alternative is better than another based on a single run since one run could represent an extreme case.

The bottom line is that a simulation that utilizes random events must be either run for a long period of time or many times (i.e., for many repetitions or replications). This is the only way to obtain data that can be used to make an accurate decision. Statistical methods can be used to determine the length of a simulation run or the number of times a simulation model needs to be run to ensure a specific level of confidence in the estimate obtained from the simulation model.

Figure 3.1 illustrates how simulation models are used. Once the value of the decision variables are set, the model is run and output is obtained. The user then changes the decision variables (based on the output from the initial run as well as the specified system performance criteria) and re-runs the model. The user continues this iterative process until a satisfactory system configuration is realized.

Almost all of the simulation models in this book include some degree of randomness. An option to use the same random number streams was selected for each example so that variable changes could be more easily compared; however, to maintain focus on the use of simulation by the Occasional User, only a single or a few runs will be requested in order to answer the questions. It needs to be stressed that this is not meant to downplay the importance of the statistical analysis, which is a critical part of any simulation project; however, it is common for Occasional Users to simply run the simulation and collect data without regard to the underlying statistics. This is because he or she typically uses a model developed by someone else, and the experimental conditions are pre-set within the model execution environment. Simulation models are commonly embedded in decision-support systems that free the user from connecting to required data sets, formatting results, and setting up experiments.

Two chapters in this book deal with statistical analysis—one for input data analysis (Chapter 9) and one for analyzing simulation output (Chapter 10). The output analysis chapter includes a discussion on the use of the Experimenter functionality included in *FlexSim*. The Experimenter facilitates carrying out a number of simulation runs and performing statistical analyses.

Section 3-3 Examples and exercises

This section provides six simulation models that illustrate a wide range of problems that are typically addressed by using simulation modeling and analysis. Through the use of models, the user adjusts

Statistical analysis is critical for all simulation projects dealing with random events.

1. capital investment amounts and resource staffing levels to balance customer service level and cost (operators and cars on a roller coaster);
2. resource staffing levels at different times during the day to assess customer service levels (checkout stations in a supermarket);
3. arrival sequence, number, and timing of transport units in a distribution system in order to reduce work-in-process inventory (space station);
4. production rate and batch size to improve throughput (bottling plant);
5. investment alternatives to improve productivity (electronics assembly);
6. production rate and schedule to increase throughput and reduce material loss (pie production).

The examples focus on system operation and improvement issues and not on model building. The intent is for the user to play the role of the Occasional User. All of the models use a custom interface, like the one shown in Figure 3.2, in order to minimize the user's interaction with modeling details or the simulation software. The user interacts with the model in a decision support role.

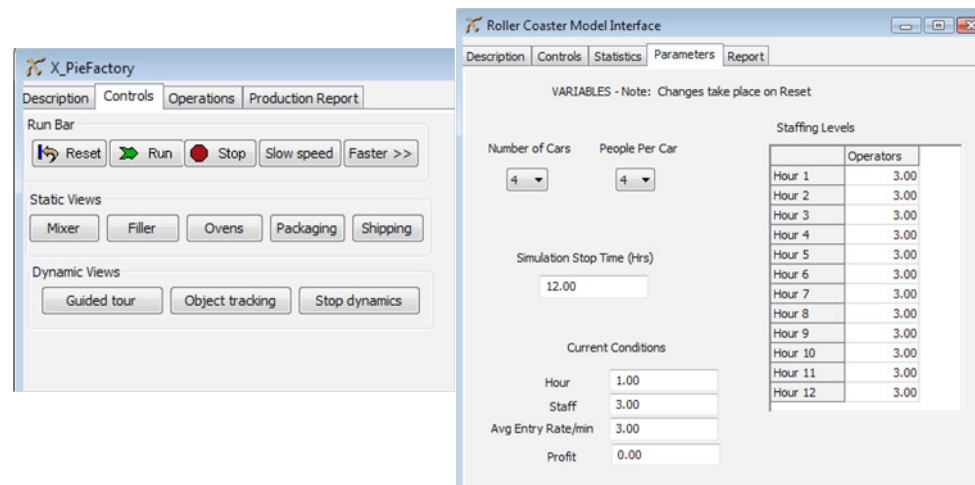


Figure 3.2 Examples of custom interface for example models

The models are all developed in *FlexSim*, but they could have been developed in any simulation software. They do not require any knowledge of the *FlexSim* software other than to open *FlexSim* first and then select to open a saved model. The simulation model and a user interface are displayed when the file is opened.

The custom interface varies for each example but has a similar structure throughout. The interface has tabs that relate to specific actions the user may want to perform. For example, the Controls tab, as shown in Figure 3.3, provides a conve-

nient way to reset, start, and stop the simulation run, as well as to change speed and view the model running from different perspectives or views.

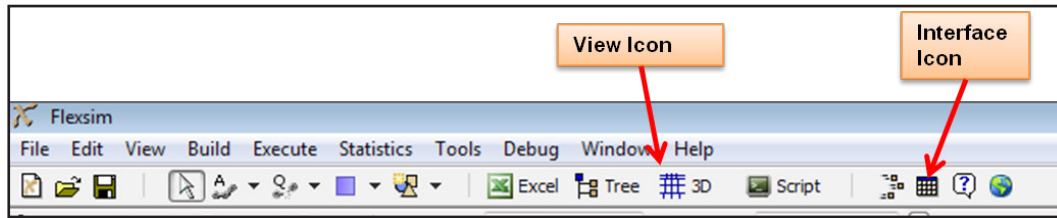


Figure 3.3 *FlexSim*'s upper tool bar

Other tabs contain the decision variables that can be changed by the user. The Statistics tab contains statistical summaries of key metrics, while the Report tab contains specific metrics collected during the run—usually in a table format. Table values can be copied and pasted into *Excel* for additional analysis. The total elapsed time is given at the bottom of the interface.

If the user interface or model view is inadvertently closed, they may be opened by selecting the appropriate icons on the upper control bar as shown in Figure 3.3. After selecting the View and Interface icons, select one of the views on the Control tab of the interface.

While simulations can be run from the interface, additional details of the *FlexSim* application environment, including moving around the 3D screen, are in the Appendix.

Exercise 3-1 Coasting around

Background

As the manager of the “Super Rocket” roller coaster ride at the *Grand Bay* amusement park, you are responsible for the day-to-day operation and financial performance of the coaster. Your parent company owns the

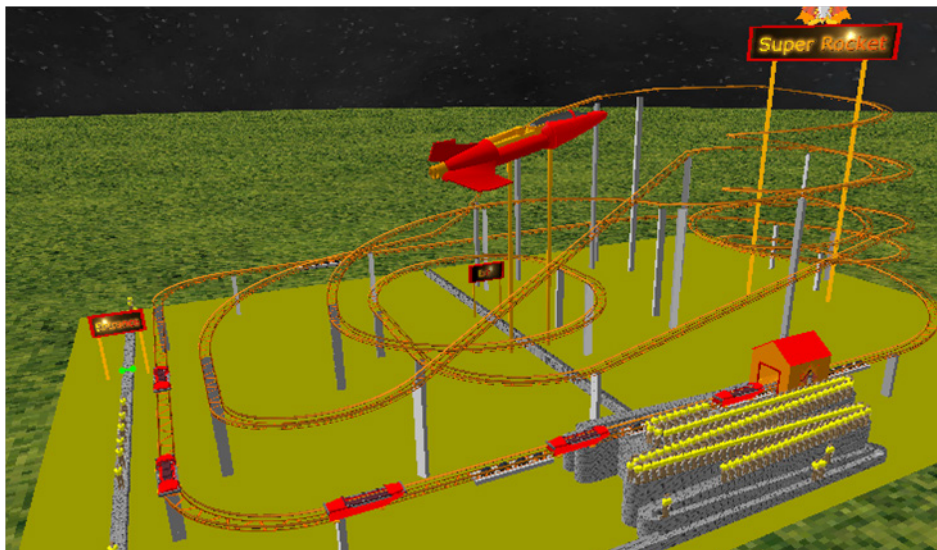


Figure 3.4 Super Rocket roller coaster

coaster and leases the space from the park. The coaster at this park is only three years old and is the lowest in financial performance of all the parent company's assets.

This year you, the manager, have to show improved financials for the ride or find another job. There are many trade-offs to consider. You are in competition with all the other rides at the park. You get paid by the park based on the number of riders served. Customers may love the coaster, but if they have to wait in line too long, they will walk away or not bother to ride again that day. Potential customers who see long lines will opt for other rides with shorter lines. Most choices to increase riders involve increased costs of equipment or staffing.

Your brother-in-law happens to be familiar with simulation methods and offered to help. You explain your problem and provide base data about the operation. You are appreciative of the results he gave you and decide to run the simulation yourself.

Problem statement

How can the profitability of the coaster be improved during the operating season?

Coaster operating data

Riders enter the coaster line at different rates during the 12-hour operating day. Operating data show the rates per minute to have a Poisson distribution; their means are provided in the table in Figure 3.5.

A minimum of three workers are needed to operate the coaster. One person takes tickets, one supervises loading the cars, and one supervises unloading the cars. Extra people can be added. The first additional person helps to unload the cars, the next helps to load the cars. In each case the unloading or loading times are reduced.

The coaster is currently operating with a base of four cars, each with a capacity of four riders. New cars can be added to the base with the depreciation costs added to the operating expense.

Rider capacity of the cars can be increased with a retrofit paid for through a depreciation charge. Other operating costs include depreciation costs for the present cars, hourly wages, and general maintenance costs. A summary of the operating costs are provided in Figure 3.6.

The following options are available to increase financial performance:

- Increase number of cars
- Increase car capacity
- Increase staffing level by hour

	Riders/Min
Hour 1	3
Hour 2	4
Hour 3	5.5
Hour 4	7.5
Hour 5	8
Hour 6	9
Hour 7	10
Hour 8	10
Hour 9	8
Hour 10	5
Hour 11	4
Hour 12	2.5

Figure 3.5 Average rider entry rate

Expected results

Provide an operating plan that includes staffing levels. Determine if there is sufficient return to request additional of cars.

Modeling and analysis issues

- How long should the simulation run? The simulation stop time is set for 12 hours; selecting the run button will continue the simulation from the stopped point.
- What variables should have the most impact?
- Is there a plan for trying different changes?
- What is most important about the financial report?
- While financial results are the primary metrics, do you feel long wait times will reduce future riders?

	Financials
CreditPerPerson	\$ 1.00
Wage/Hr	\$ (6.50)
OldCarDep/Hr	\$ (3.00)
NewCarDep/Hr	\$ (7.50)
Maint/car/hr	\$ (25.00)
CarUpgradeDep/Hr	\$ (1.10)

Figure 3.6 Operating Costs

Exercise 3-2 Farm Pride

Background

As regional manager of the *Farm Pride* chain of local supermarkets, you are considering expanding operations by building a new market in the Snyder's Point area. You've found an ideal location that should attract middle class families who favor specialty foods along with prepared meals, and the basic necessities. The contractor wants to know design details, and your HR department wants to know staffing levels. You've asked your business development group to assist in the design. They have a simulation based on your other stores' layouts to help with the design, but you have to perform the analysis yourself.

Problem statement

Based on the general trends of shoppers in the proposed market area, what staffing levels should you anticipate? Identify general rules for the number of registers open based on the number of people entering the store.

Operating data

Shoppers enter the model with a randomly generated shopping list and pick up all their groceries from the appropriate shelves. Some shoppers go to the deli, where

they choose their meat and have it sliced by the worker. When they are finished shopping, they travel to the shortest checkout line to pay for their groceries.



Figure 3.7 Farm Pride Supermarket

There are three definable periods of time for shopping:

- The high volume period, when the most shoppers with the longest shopping lists are in the store.
- Lunch time, when business is less brisk but the demand on the deli counter is higher.
- Off-hours, when the least number of shoppers arrive.

The various periods are characterized by the shopper entry rate, the number of items they shop for, and the percent of their shopping list that might contain deli orders.

Expected results

Identify staffing levels at the checkout area for various times of the day.

Modeling and analysis issues

- What period of time should the simulation run?
- Is the time-in-store metric valid if it includes the time in the checkout line?
- What other metrics may be appropriate?

Exercise 3-3 Martian Transfer Station

Background

The space exploration task force on Mars has seven laboratory stations. Supplies are brought in on cargo rockets and then transferred to their destination by surface vehicles. Pallets are recycled and brought to the transfer station where two robotic cranes separate the incoming supplies. When a pallet is complete for a station, a guided vehicle picks it up and carries it to the station. The transfer station is operated by two locals.

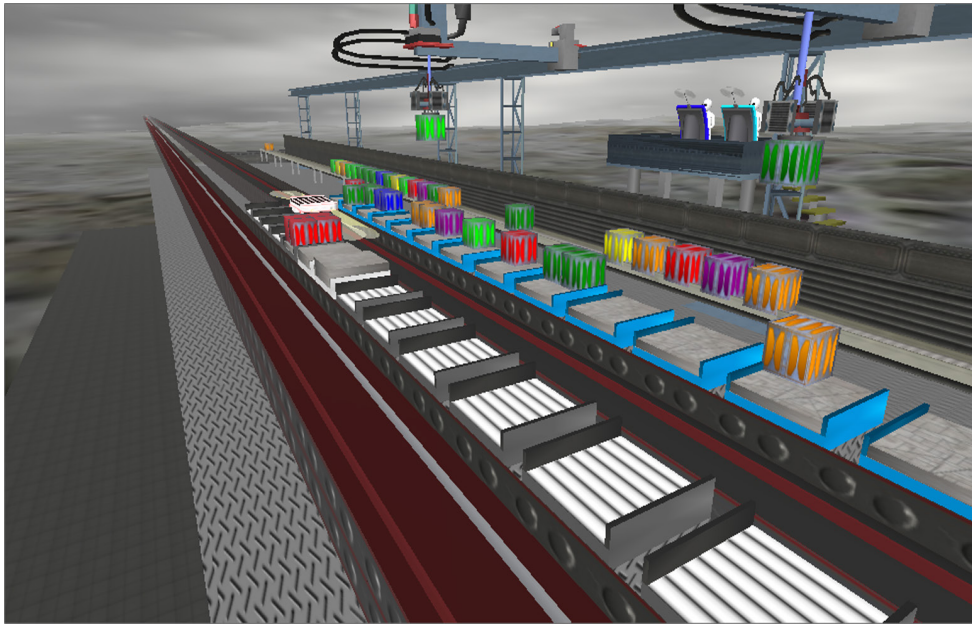


Figure 3.8 Martian Transfer Station

Problem statement

The stations are receiving increasing amounts of supplies—some of them with time-critical contents. Can the work-in-process inventory be reduced by changing operational parameters?

Operating data

The transfer station has two working cranes. Empty pallets are delivered by a transporter vehicle and transferred to the loading station with a shuttle. A crane will only start loading the next box of supplies if there is a pallet in the appropriate transfer station.

Each pallet contains four crates. When a pallet is filled, it is moved to the distribution side by the shuttle and a call is made for pickup. When the transporter that was called stops, the full pallet is loaded and taken to the experiment station. Transporters already in-route will not stop for the pallet.

Only a certain number of transporters are available on the planet. They are sent out at specific time intervals. After going through the transfer station they continue

on to one of the seven lab stations. After depositing any load, they continue to a transportation center, where they are brought back as a group to the sending station to once again be sent to the transfer station.

The supply rocket is normally packed randomly on earth as supplies are brought to the spaceport. Requests have been made to stage the rocket load first, and then group them on the rocket by sequence, with a number of boxes grouped by final destination.

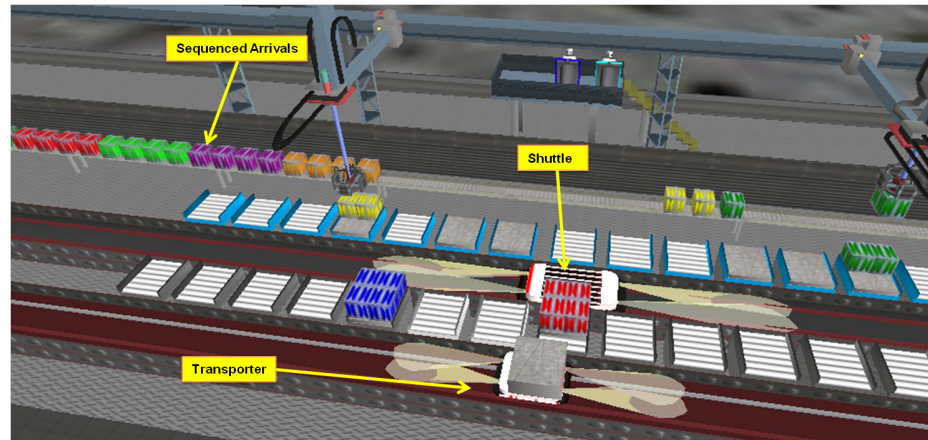


Figure 3.9 Details of martian transfer station

One time unit is one Martian second. As on Earth, there are 60 Martian seconds per Martian minute and 60 Martian minutes per Martian hour.

The operating variables that can be changed include: V1 – the number of transporters; V2 – the time between transporter departures; V3 – the number of empty pallets per load; V4 – selection of either random or sequenced incoming loads; and V5 – if sequenced, the number of crates grouped together (pattern quantity).

The operating rules can be expressed as follows:

- A group of transporters (V1) leave the sending station at a fixed interval (V2).
- If empty pallets are needed or a pallet is in place and ready for pickup, a request is sent to the sending station and the next transporter is given the assignment. A transporter can have only one assignment at a time
- Only one transporter can be at the transfer station at a time—others will bypass the station
- When all transports in the group arrive at the transportation center they are returned to the sending station. Return transit time for a group of transporters is 80 Martian sec.
- Crates from the supply ship arrive either in a random sequence or in a grouping of a number of crates (V5) for each destination.

Expected results

- Run the simulation for a period of time and observe the system dynamics. How do the cranes work to position crates on pallets?
- Devise a plan to maximize the distribution of materials to the laboratory stations. What are the tradeoffs with cost per crate?

Modeling and analysis issues

- What is the impact of the arrival sequence?
- Can you tell by observation what logic is employed in moving material from the space ship to the pallet layout?

Exercise 3-4 Slime Inc.**Background**

As the new engineer working for Slime, Inc., you've been asked to familiarize yourself with the company's operations. This plant only bottles the green variety of slime and ranks in the middle of all plants in performance. Overall plant performance across package sizes is a key metric, but performance of packing the popular 12-pack of slime is critical from a market perspective. You've found a simulation of the plant that was built last year and decided to try it as a way of understanding the operation.

Marketing is pushing the plant to agree to higher speeds and smaller tray pack units because people can only handle so much slime at a time.

As an incentive, bonus points are awarded based on meeting production targets and result in additional pay at the end of the fiscal year.

Problem statement

What are the best operating conditions to meet marketing and plant needs?

Operating data

Starting from raw plastic, the bottles are created with an injection molding machine. The plastic forms are puffed into the bottle shape by a blow molder. The empty bottles are put into baskets that transfer them to an un-scrambler, which in turn lines them up to go into the filler lines. There are two filler lines that clean, fill, cap, and label the bottles. The two lines join at the single tray packer. Trays are conveyed to a case closer and brought to an operator who loads a hand truck to put them into waiting trucks.

One operator is responsible for the molding and unscrambling operations. This operator unloads the baskets of clean bottles and places the basket back on the return conveyor. The operator also must attend to any downtime on the equipment

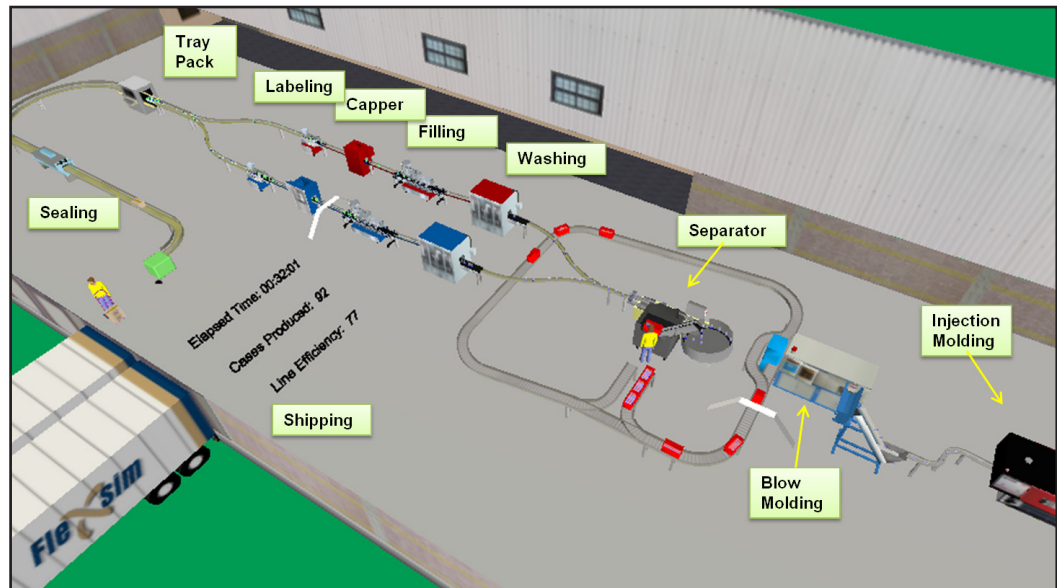


Figure 3.10 Slime Inc. bottling line

in the area. The second operator carries cases to the trucks and fixes any problems with the filling and case packing operations. Operators are concerned about their workloads under the higher production rates.

Once a truck is filled, the facility's doors will close and re-open only when another truck has arrived.

The plant can operate at molding speeds between 45 and 85 bottles per minute. At higher speeds the machines become less reliable and jam more often. This requires the operators to stop their production related work and fix the equipment.

The tray packer can handle 12, 16, and 24 count trays. Production standards for the plant were set based on the 24 count trays when the plant was built. At a given bottle rate, more 12 count trays are produced, thus requiring the second operator to make more trips into the trucks carrying 5 cartons per load.

Plant efficiency is rated at better than 97% based on their standard 24 count tray and 55 bottles per minute. Lower efficiencies will result in lower bonus points.

Expected results

Considering changes in bottles per case and the production rate, prepare a recommendation for the next engineering group meeting indicating what you've found out about the operations and what might be done to improve performance and meet marketing needs.

Modeling and analysis issues

- For what period of time should the simulation run?
- Does the production or utilization report give any indication of a bottleneck?

- Note what happens to operations as line speed increases.
- How busy do the operators appear to be?
- Is there a tradeoff between the plant's operating performance metric and the need for smaller sizes?

Exercise 3-5 Danson Electronics

Background

The Danson Electronics plant in Pembroke has been assembling high quality DVD players for two years. To improve their operations, they instituted a continuous improvement program with teams active on the plant floor.

Operating data

The plant is divided into three buildings. In the first, double circuit boards are populated with components and then sent to a wave soldering machine. After inspection they are grouped onto a tray and manually transferred to the next building.

In the second building, the double boards are separated using a router, and the individual boards are then put through a series of tests. After testing, the boards are coated and put in racks to dry. Dry boards are manually transported to the third building for final assembly.

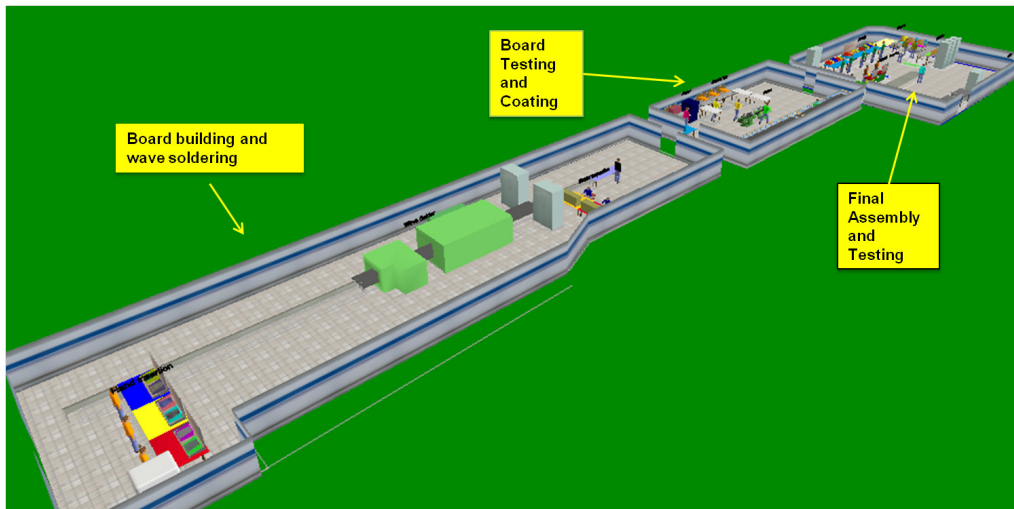


Figure 3.11 Danson Electronics' DVD production facility

In the assembly building the boards go through a series of assembly steps and tests. The assembled units are placed on a table and moved to an aging box. After sitting in the aging box, the units are tested again before moving on to shipping.

Four proposals have been put forth to improve plant productivity:

- The first building team notices that boards are backed up through the wave soldering machine. They attribute the backup to the fact that the router

operator only has another batch of boards brought over when the current tray was empty. They want a surge added before the router so that boards can be moved more effectively.

- The second building team wants to purchase a new router that would decrease the routing time by 20%.
- The third building team wants new assembly machines to increase their throughput by 20%.
- The union wants another person hired for the second building to help move boards between buildings 2 and 3.

Problem statement

Which proposal should be adopted, or should all of the proposals be considered with a priority established. What else should be done?

Expected results

Suggest how the improvement team recommendations should be handled.

Modeling and analysis issues

- What should the simulation run time be?
- Where do backups occur?
- What priority should be given to the suggestions?
- What improvement level would be considered good?
- What is most important: total production, WIP, or quality numbers?

Exercise 3-6 Grandma's Pie Emporium

Background

Grandma's homemade pies became so popular that the company set up a simple packaging system. At one time they offered up to three pies in a box; however, the sequencing of products through the plant became more complex. Sales have increased and more demands are being placed on the plant staff.

Grandma loves her pies but loves profit as well. The plant manager has complained that the schedules given to them do not optimize production and often require overtime. They also need to increase their production rate.

Problem statement

What is the best scheduling approach to operate the line and fill orders while keeping the most customers happy and shipping the maximum number of pies in the least amount of time?

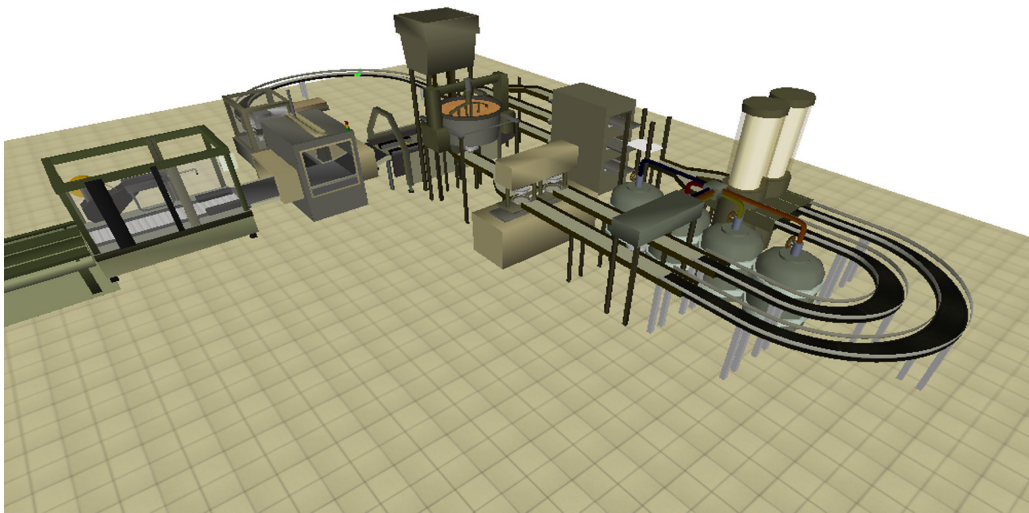


Figure 3.12 Grandma's pie factory

Operating data

The plant runs three shifts, 24 hours a day. Currently four types of pies are produced on the main line: apple, blueberry, strawberry, and custard. The plant has two filler stations that can produce the pies but require a clean out time between products, as shown in the following table (times are in minutes).

From/to	Apple	Blueberry	Strawberry	Custard
Apple	2.00	6.67	6.67	6.67
Blueberry	10.00	2.00	6.67	10.00
Strawberry	6.67	6.67	2.00	10.00
Custard	6.67	6.67	6.67	2.00

From the filler stations, the pies are brought by conveyor to the topping section where the top crust is applied. The completed pies go through a continuous oven where they are cooked to a golden brown.

The cooked pies are packed two to a box and sent over three conveyors to a palletizer. The palletizer combines 18 boxes to a pallet. The pallet is then wrapped and set out for shipping.

<u>Number of pies</u>	
Blueberry	504
Custard	576
Apple	540
Strawberry	468
Custard	432
Apple	648
Blueberry	612
Strawberry	432

The sales department sent the following shift schedule to the operations team. They expect the pies to be produced in the order received.

The plant was designed to operate at six pies per minute. Corporate engineering feels that the rate could be increased to as high as ten pies per minute. The operators collected data trying to run the line at various speeds and found that the filler reliability decreased at higher speeds. The graph in Figure 3.13 shows filler efficiency at various pan speeds.

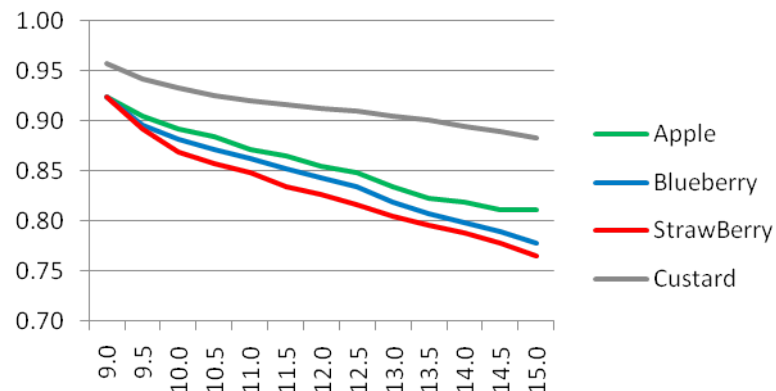


Figure 3.13 Filler reliability vs. pan rate

Expected results

Determine scheduling rules that will improve profit; that is, modify the production rate and production schedule. The simulation will stop when the production run is complete.

Modeling and analysis issues

- How long should the simulation run?
- What general scheduling logic might be used to optimize the schedule?
- What is the shortest time to complete the orders?
- Does the amount of waste impact the results?
- How does the result affect marketing claims of orders being run when received?

Section 3-4 The next step

This chapter emphasizes how an Occasional User can use simulation as a tool to make more informed decisions. For the simulation to be useful, the user must trust the way the simulation was developed, especially if the simulation values cannot easily be validated. There is a methodology to building simulation models and understanding that methodology is important for both the Occasional User and those who build the simulation model.

There are three main steps involved in developing and using a simulation to support decision making:

- Specifying the objectives and scope of what should be simulated.
- Building the simulation model.
- Experimenting with the model and analyzing the results.

The Occasional User is always involved in the first step as they may be the owner of the issue being simulated. The Occasional User may also be involved in studying the results—as the previous exercises illustrated. Following a methodology for the three steps is important for the communication that must be maintained between all that are involved in the simulation project.

The next chapters indicate the interpersonal and technical issues involved with using simulation as an analysis tool. They define how a simulation project should be conducted. The chapters following that methodology discuss the skills and knowledge that are needed to successfully build simulation models and apply the appropriate methods to analyze the results.

Chapter 3 - Review questions

1. Identify three “nuggets” – the things you found to be the most interesting or most important – in the chapter.
2. Discuss why simulation is considered to be an applied technology.
3. Select an operations system and identify some controllable and uncontrollable variables; identify which of those might be considered random.
4. Discuss why stochastic simulations need to be run more than one time.
5. Describe what knowledge is needed for an Occasional User to use simulation as an effective decision-making tool.
6. Describe the three main steps involved in developing and using simulation to support decision making.

Chapter

4

Professional Practice of Simulation

It is of paramount importance to have the results and recommendations derived from a simulation model accepted and implemented. After all, the power of simulation and all of the effort expended to build models and conduct analyses is lost if it is not used. The most technically elegant model is of no value if it does not support decision making in a timely manner. This chapter addresses the non-technical aspects of simulation modeling and analysis, including the importance of user confidence, understanding that simulations have a life cycle and involve a definitive process, and the many roles that must be fulfilled in order for simulation to be applied successfully. This chapter also discusses model-based decision-support systems and concludes with a set of recommended success factors that are critical in simulation projects.

Section 4-1 Confidence

When a person has a serious medical condition and a new technology is available, confidence is a major factor in choosing whether or not to use the new approach. Not only confidence in the technology itself, but also in the person using the technology and in the process by which the technology is applied. Of course, a person who is desperate for help may choose the new procedure even without total confidence. Decisions to use simulation can be very similar. Simulation is a tool for analyzing and solving problems associated with operations systems. As with the medical example, it is a technology that may be new and unfamiliar to those seeking help in solving a problem. Although an intense desire to find an answer to a problem may outweigh other considerations, confidence is critical to the successful use of simulation and to the professional success of the person using it.

Most of this book deals with the technical issues associated with simulation modeling and analysis (SMA). Of course, a technically sound model and analysis

Credibility, built on confidence, is key to obtaining acceptance for the use of simulation.

are necessary conditions for project success; however, that alone is not sufficient to guarantee success. For example, an elegant model may address the wrong problem, or the key stakeholders may have little confidence in the model, the person doing the analysis, or the results it generates. Any of these issues could cause the simulation project to be deemed a failure. It often takes a considerable amount of time and effort to get an organization to accept the use of simulation, yet that acceptance can be lost very quickly with a single, poorly run simulation-based analysis. It is necessary, therefore, that significant attention be paid to the non-technical issues involved in a simulation study and the way confidence is achieved.

Confidence in the technology

The first two chapters provided numerous examples of how simulation is used to successfully solve problems. For some, seeing examples from a single source may not be sufficient enough to be convinced of its effectiveness. They may require additional examples from published papers, conference proceedings, or roundtable discussions with others in their industry. Senior managers may suddenly be interested in simulation based on a presentation at an executive conference. Building confidence in the technology requires educating others by examples.

Confidence in the person

Since simulation is a tool for solving problems, the primary way confidence is built in a person who uses simulation is through their problem solving ability. When that ability is demonstrated, with or without the use of simulation, people will have confidence in that person. For a person new to an organization, being aggressive and eager to solve problems will raise management's confidence in you. Being a successful problem solver means focusing on listening to people, finding the underlying issues, and responding in a timely manner. Even with a powerful tool such as simulation, the emphasis must be on solving the problem and not on the technology.

Confidence in the process

While simulation may, at times, be used solely by the person solving the problem as an aid to their analysis, it is more commonly used by a team as an integral part of the problem solving process on a large issue. In these cases it is critical that everyone involved in the process clearly understands the methodology and takes an active role in the process. Only when all are actively involved will there be confidence in the results of the simulation analysis. With such confidence, it will be easier to gain the support of those outside the simulation project who may be required to accept the simulation results.

Building confidence often begins with small steps. A good way to start is by observation and analysis, which can lead to the identification and resolution of a

Building on success and a proven track record can establish confidence.

problem. The entire confidence building process can be summarized in the following statement (first put forth in print by Sir Arthur Helps in *Realmath* in 1868):

Nothing succeeds like success.

In this chapter, we will define an SMA life cycle and process, introduce the use of decision support systems, define the roles that must be filled in an SMA project, and offer some key non-technical practices that can improve the chances that a simulation project is a success and delivers the most value. This chapter provides the outline of the process while the next chapter will provide the details of actually carrying out the project.

Whether the problem analysis is simple enough for one person or large enough that a team is involved, the lifecycle and process remain the same.

Section 4-2 The Simulation Modeling and Analysis (SMA) Life Cycle

When simulation is used to address a problem, it is not an event or an activity; it is a project. As such, simulation studies, like all projects, have a life cycle, similar to the one shown in Figure 4.1. Key phases in the life cycle can be described as follows: determining the objective(s), developing the model(s), performing analyses, implementing a change, assessing performance, and ultimately realizing results.

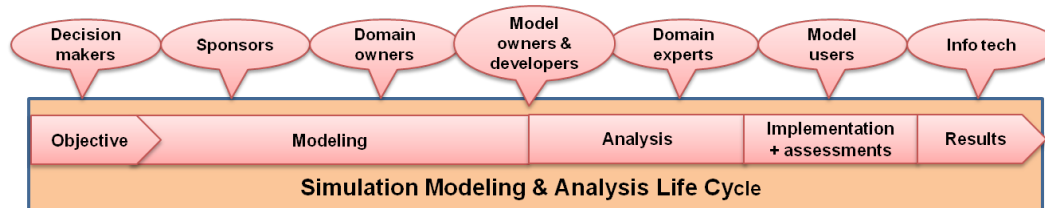


Figure 4.1 The simulation modeling and analysis (SMA) life cycle

Normally, a project would conclude with implementing the suggested changes, assessing how well the project was carried out, and documenting the lessons learned. We include the results phase to indicate the outcome of a simulation may be relatively short-lived, as is the case when simulation is used to support an investment decision. Or, the results may continue for a long time with recurring benefits accruing from the use of a simulation model, as is the case when a simulation model is used routinely to enhance planning or operations. It is important to think about results right from the beginning, as Stephen Covey posits, “Begin with the end in mind.”¹ Thinking about the outcome helps to set the objective, define the requirements, and assess the level of model detail that is needed to adequately address the problem.

Figure 4.1 also identifies the roles of stakeholders that typically participate in a simulation project. While some of the roles may be carried out by the same person, it is important to understand all of the various roles in the project. For example, if simulation is used to make production planning decisions, the vice president of manufacturing may be the ultimate decision maker (on what to produce or where to

A consistent approach is important regardless of the simulation size.

It is important to be aware of all the people involved in a simulation project.

produce it) as well as the sponsor (the one providing funds for the study and development); the production manager may be the domain owner; the production workers and technical staff may be the domain experts; the model owners may be the industrial engineering department. The actual development may be shared between internal engineers and external consultants, and the model users may be production planners. Much of the data may need to be extracted from company databases, and thus the information technology must provide access and possible database support.

The example above illustrates that most simulation studies are multi-disciplinary, cross-functional projects and must be managed that way; focusing only on the modeling phase will likely lead to an unsuccessful project.

Section 4-3 The modeling and analysis process

While each simulation study is a separate project, each involves a fairly standard set of interrelated activities. A critical first step is to clearly articulate the objectives of the simulation study and to develop a project plan. The project plan is basically a more detailed breakdown of the activities shown in Figure 4.2. The plan will typically indicate key milestones, provide estimates of the duration of each activity, and assign responsibility as to who leads and performs each activity. The plan itself is oftentimes summarized graphically in a Gantt chart. Figure 4.2 provides an operational view of simulation activities, similar to the way they would be represented in a project plan.

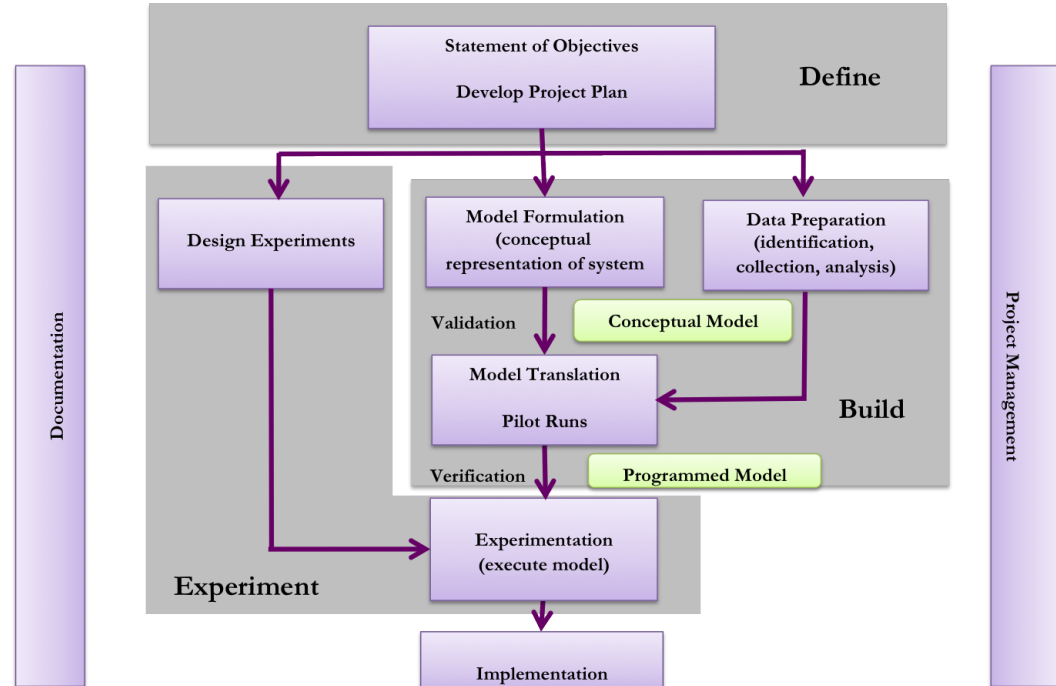


Figure 4.2 The SMA process

The main portion of the SMA process can proceed along three parallel paths. The central path is model related and associated with its development. The two related

Building a simulation has many inter-active elements.

paths are output or analysis related and input or data related respectively. Obviously as the project progresses, there needs to be good communication among the people performing the tasks along these paths—while the tasks may be conducted in parallel, they are not independent. Also, they typically involve different stakeholders.

The modeling path begins with the development of a conceptual model, typically a high-level diagram that captures the key elements in the system and their relationships. Its purpose is to document how the system works, provide an understanding of how detailed the model needs to be to meet the analysis objectives, identify data needs, and uncover possible modeling challenges.

The conceptual model is a key means of validating the simulation model and addressing whether the *right* model is being built. Validation determines whether the model is a meaningful and accurate representation of the real system. The model's accuracy is closely linked to how the model will be used.

The programmed model (the one that is used for experimentation) is a translation of the conceptual model into computer-executable form. Prior to experimentation, the programmed model must be verified; that is, it must be determined that the model is working as intended. In this step, the focus is on building the model *right*.

There are two activity boxes on either side of the main process flow in Figure 4.2. The one on the left—documentation—indicates that the model and project need to be documented throughout the process and not delayed until the end. Most people don't like developing documentation, but it is extremely important for future modifications to the model, as well as explaining rationale, identifying and justifying assumptions, providing details of model logic and analysis, and capturing lessons learned. The activity box on the right side of the main process flow—project management—emphasizes the need to use good project management practices throughout the study.

Figure 4.3 focuses more on the technical aspects of a simulation study. The technical focus is simulation model development, the details of which make up the remainder of this book.

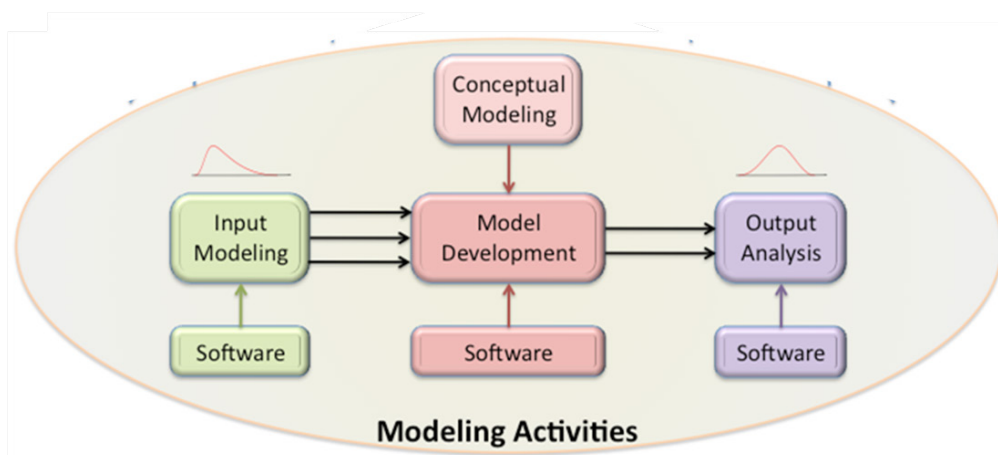


Figure 4.3 Primary technical aspects of simulation

Models are developed through commercial simulation software packages available in the marketplace. This book's focus is obviously on making the translation through *FlexSim*. Ideally, the computer models are based on solid, software-neutral conceptual models—that depict the key elements in a system and their relationships.

Since simulation models are data driven, considerable time is dedicated to input modeling, such as characterizing input data as probability distributions. As Figure 4.3 indicates, these activities are supported through software as well. The software can be special purpose, such as *ExpertFit* or *Stat::Fit*, or general purpose, such as statistical analysis packages like *Minitab*, *SAS*, or *SPSS*.

The main reasons to build models are to conduct analyses, increase understanding, and obtain results; therefore, considerable effort is associated with analyzing output obtained from simulation models. These activities are also supported by software. All of the major commercial simulation software packages contain means to capture and analyze the data output from simulation models. Depending on the analysis needs, these built-in capabilities can be supplemented with special purpose software, such as those mentioned above, for input modeling. In addition, if the simulation model is used to drive an optimizer, then special purpose software, such as *OptQuest*, may be used.

Section 4-4 Roles in a SMA project

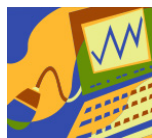
Too often the roles of individuals in an SMA project are considered solely within the nomenclature associated with the project activities described above. As a result, the only roles that are explicitly addressed are those that deal with model construction, and maybe analysis; however, there are many roles that can be valuable to a successful SMA project. Of course, some individuals may fulfill a variety of roles. Those roles are defined below.



Developer: Develops and constructs the simulation model and/or model support software, such as database connectivity, user interfaces, decision support systems, etc.



Designer: Establishes the model and/or decision support system architecture; this role is closely related to that of developer.



Analyst: Designs (e.g., defines what variables need to be changed in each scenario and their value, determines run length, warm-up period, number of replications, etc.), conducts, and summarizes experiments.

Simulation projects involve many tasks that may easily be overlooked.



Researcher: Conducts research early in a project to see if similar issues have already been addressed elsewhere and what can be learned from previous work.



Investigator: Probes and “digs” to clearly define the processes and investigate all the available data in order to ensure the system is clearly understood and adequately represented.



Educator: Helps stakeholders involved in a simulation project understand enough about SMA to be an effective contributor; this is necessary because stakeholders have diverse backgrounds and often-times have limited expertise in simulation.



Implementer/user: Converts the results from the simulation into actions and changes, this includes planning, obtaining buy-in, and follow-up. If results from the SMA project are not implemented, then it has added little value to the organization and can in fact be considered a liability

One way to clearly articulate who is responsible for performing what role(s) is to construct a responsibility matrix early in the project. As shown below, a table is constructed that identifies key project activities associated with each role and identifies the specific project member who is responsible. Oftentimes, more than one person is responsible for different activities and roles. When this is the case, one individual needs to be indicated as the “lead” so that he or she can coordinate and properly manage the work.

Role	Activities	Responsibility

As Figure 4.4 illustrates, SMA projects involve two types of knowledge: domain knowledge and simulation knowledge. Domain knowledge is specific to the system being modeled and is usually provided by the company where the work is being done. Simulation knowledge is acquired through books like this one, courses, and practice. The figure also illustrates that SMA projects must have organizational support and must be managed as projects.

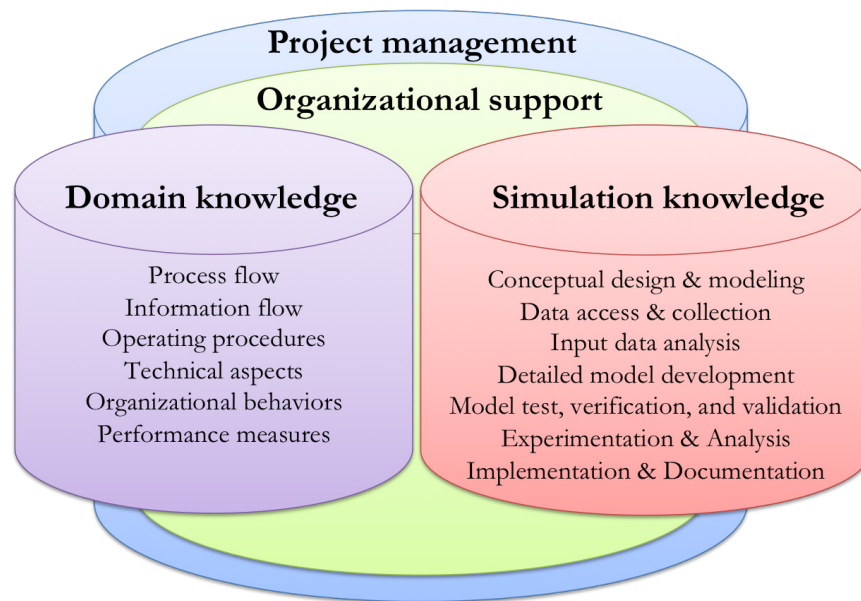


Figure 4.4 Knowledge needed in a simulation project

Section 4-5 Model-based decision-support systems

Most applications of SMA are to support decisions. These may be one-time decisions, such as selecting the best layout for a new facility, choosing the best policies or rules to use under varying conditions, exploring whether a proposed change is feasible, etc. On the other hand, the decisions may be ongoing, such as selecting where to produce certain products, deciding whether an order can meet the desired delivery time, identifying the best sequence to process a set of products, etc.

Oftentimes in one-time analyses, simulation practitioners or advanced users are involved and are the ones interacting with the simulation software. In this case, the user is knowledgeable about simulation and comfortable with the software; however, that is not usually the case when simulation is used on a regular basis to make decisions. In this case, it is usually impractical to have the user be very knowledgeable about simulation; therefore, the user needs assistance in interacting with the simulation model.

This assistance involves providing, at a minimum, a friendly graphical user interface (GUI) that enables the user to input data easily (e.g., through a form or spreadsheet), and obtain information in a familiar format (e.g., graphs or spreadsheets). The assistance can be much more extensive, as illustrated in Figure 4.5. In this case, the decision support system (DSS) not only provides interfaces to the user, but to schedule, product, and process databases as well. It also interfaces with analysis routines and algorithms, such as optimization, in which the simulation model is used to provide feedback to an optimizer in order to help guide its search for better solutions.

A key role of simulations is to assist in making sound decisions based on analysis and results.

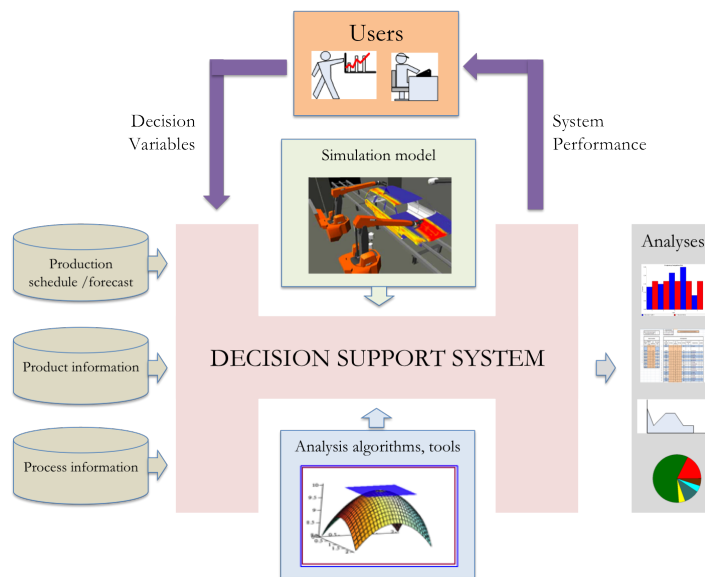
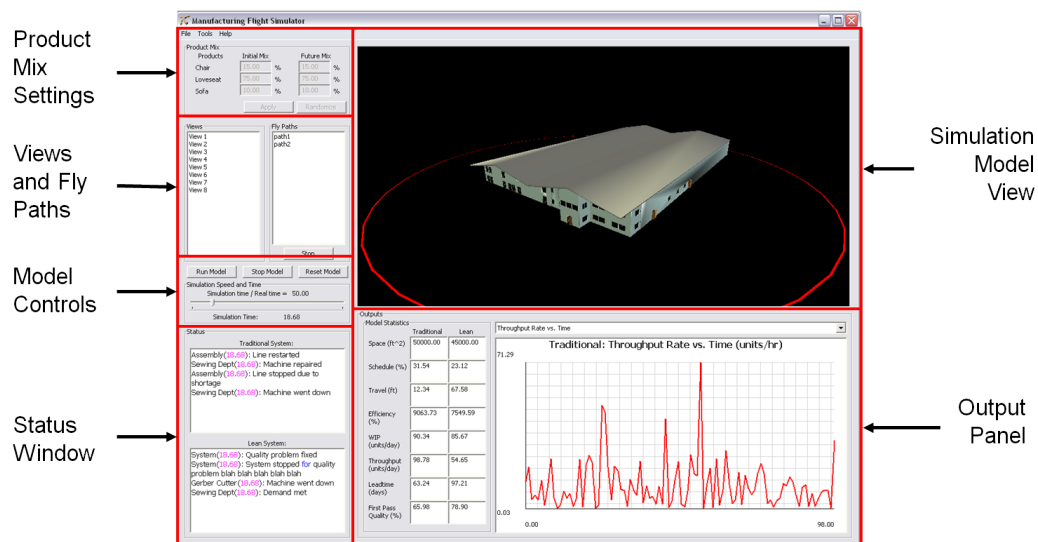


Figure 4.5 Components of a simulation-based decision support system

An example of a decision support system is shown in Figure 4.6.



Source: "A Decision Support System for Rapid Comparison of Lean-to-Batch Systems," Center for Advanced Vehicular Systems – Extension, Mississippi State University, 2007.

Figure 4.6 Example simulation-based decision support system

The user

- inputs data on the left side of the interface;
- observes output from the model via a graph and table at the bottom of the interface; and
- views the model execution in the center window.

All of these activities are performed in one place. This DSS was developed all within *FlexSim* using its GUI builder.

**Success factors
involve both
technical and
people skills.**

Section 4-6 Simulation modeling and analysis success factors

In summary, the following are a few key non-technical practices that can help simulation projects be a success:

- Manage the simulation modeling and analysis activities as a project.
- Define clear objectives early and get buy-in from stakeholders.
- Set realistic expectations and manage them.
- Identify early how the simulation will be used and by whom; understand the users' needs.
- Obtain management support.
- Involve stakeholders throughout the process; gain their acceptance.
- Test, validate, and verify the model.
- Understand the data involved (its availability, format, reliability, quality, etc.).
- Start simple; add complexity as needed.

References

1. Covey, Stephen R. *The 7 Habits of Highly Effective People*, New York: Free Press, 1989.

Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the three areas of confidence that are important for getting simulation to be used to support decision making in most organizations.
3. Discuss why it is a necessary, but not sufficient, condition for simulation project success to have a technically sound model and analysis.
4. Discuss the key phases of the SMA Life Cycle and describe the stakeholders involved in the life cycle.
5. Outline the key activities that need to be performed in a simulation project and discuss the relationships among the activities.
6. Define conceptual model and identify the benefits of constructing a conceptual model.
7. Discuss the differences between a conceptual and programmed model.
8. Discuss the differences between validation and verification.
9. Identify the main reasons for building models.
10. Discuss the various roles that are prevalent in a simulation project.
11. Define the two main classes of knowledge that are key in a simulation project and identify specific knowledge areas within each.
12. Discuss the types of assistance that a decision support system can provide to an occasional user.
13. Describe how to employ key non-technical practices in a simulation project to help ensure that it succeed.

Chapter

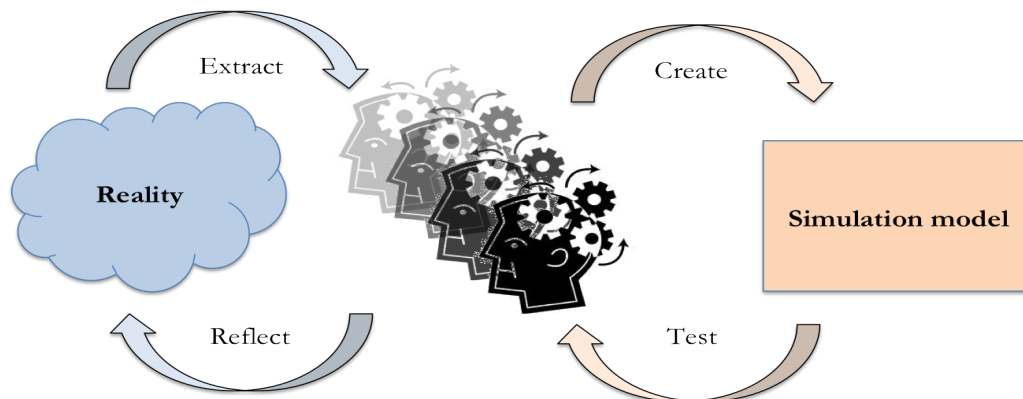
5

Managing a Simulation Project

The previous chapter provided a general outline of how the SMA process is used for analyzing and solving problems with simulation. It highlighted the people and organizational issues involved with the successful use of simulation. This chapter focuses on the technical details of carrying out the SMA process and provides tools that can be used in the process.

Section 5-1 The starting point

An organization that needs a simulation for analysis must follow a series of steps in order for the simulation to be successful. Building a simulation involves translating reality into a time-based model. As individuals look at the actual system, they develop their own conceptual model of how the system operates. Each person may see the system slightly differently or notice different characteristics—all of which may be important. Consequently, a methodology is needed to aide in communications and general understanding.



Adapted from Pidd, M. *Tools For Thinking* 2nd Ed, Chichester, England: John Wiley & Sons, 2003, p. 18.

Figure 5.1 A Simulation is a representation of stakeholders' reality

The Occasional User may not be directly involved with the actual building of the simulation but will be in a position to define what the simulation should accomplish and then possibly carry out simulation runs for analysis. Knowledge of all the steps involved with a simulation project is critical for assuring the project and results are valid.

Simulation project steps

As introduced in Chapter 4, a simulation project is a joint effort between those wanting the simulation and those developing it. The result will only have real value to the people who are involved with the work. There is a straightforward methodology for defining and carrying out a simulation; this methodology will assure that the resulting model will be of value. It requires working through the steps shown in Figure 5.2; the level of detail for each step will depend on the complexity of the system.

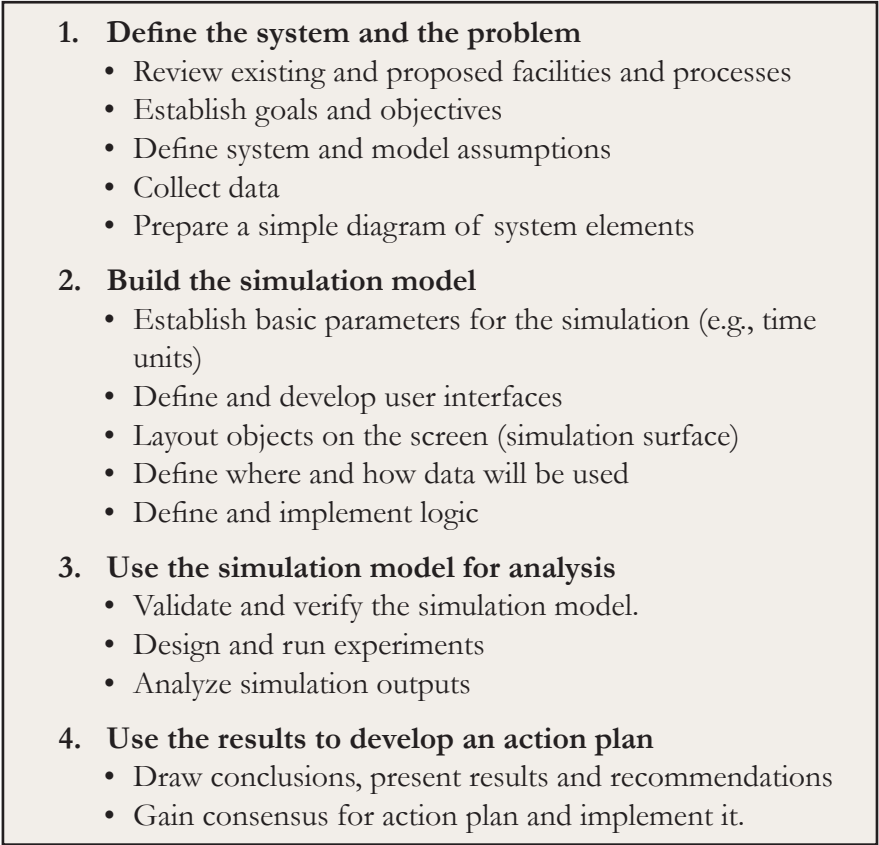
- 
- 1. Define the system and the problem**
 - Review existing and proposed facilities and processes
 - Establish goals and objectives
 - Define system and model assumptions
 - Collect data
 - Prepare a simple diagram of system elements
 - 2. Build the simulation model**
 - Establish basic parameters for the simulation (e.g., time units)
 - Define and develop user interfaces
 - Layout objects on the screen (simulation surface)
 - Define where and how data will be used
 - Define and implement logic
 - 3. Use the simulation model for analysis**
 - Validate and verify the simulation model.
 - Design and run experiments
 - Analyze simulation outputs
 - 4. Use the results to develop an action plan**
 - Draw conclusions, present results and recommendations
 - Gain consensus for action plan and implement it.

Figure 5.2 Simulation project steps

The general structure of this process follows Deming's Plan–Do–Study–Act cycle. The first phase is all about planning and understanding. The second phase involves abstracting the system's behavior into model form. The third phase concerns using the simulation model for analysis or study. The final phase is action oriented; that is, it involves gaining insight and information and deciding how to improve system performance based on various experiments conducted with the model.

Providing a concise report of results defined by the objectives helps to build credibility in simulation technology and the people who use it.

Activities performed during these steps should be documented using the Simulation Project Template found in the Appendix.

Documentation is essential for helping all the individuals working on the simulation to have the same view and understanding of the project. The project template was specifically developed for defining and managing actual simulation projects. The first item in the project template is the project name. The simulation should be given a name that has meaning to the team members and others. The simulation models should contain the name followed by the current version number (e.g., BankSim1).

Since the simulation project is normally approved and carried out for a particular purpose, a summary of the results is usually required. This executive summary is normally limited to one page or one slide and focuses on the reason for the simulation and the result of the analysis. The three main parts of the summary are as follows:

- Background (reason for the simulation and objectives)
- Results
- Conclusions and recommendations

Section 5-2 Define the system and the reason

The first phase of a simulation project involves the critical task of actually defining the project. This definition and the associated material are often referred to as the *functional specifications*. The functional specification can be the basis for requesting a quote from a third party to build the simulation.

This phase of the process involves listening to those who are responsible for the problem in question. It also means reviewing the facilities and the processes that will be represented by the proposed simulation model. This review can include a tour of an existing facility, and/or a review of design and operations documentation, especially for a proposed system if it doesn't already exist. In either case, everyone involved should have a complete and consistent understanding of the operational characteristics of the system. The level of detail at this point helps to focus the effort.

Several things need to be accomplished in this review:

- *Tour existing facility:* Even if simulation project team members work in this facility, a complete review is still helpful.
- *List system components:* During the tour, list all the system components including machines, storage locations for work in process, work tables, tool locations, or anything that is used to produce the product. Product components, parts, and assemblies should also be listed.
- *List system resources:* Also list all system resources, including conveyors, fork trucks, AGVs, maintenance personnel, machine operators, overhead cranes, robots, or any other resource that is used for production.

Documenting a simulation may seem tedious but it's the only way to ensure that the work will continue to be of value.

The functional definition for a simulation model is a communication document for managers and implementers.

Since a simulation reflects reality, knowledge of the system to be simulated is paramount.

- *Make note of operational characteristics:* As each system element is reviewed, make notes of the operational characteristics. Is there any special processing logic? How is work selected? How do equipment and people interact with each other and other elements of the system?
- *Make a list of system terminology and acronyms:* Make a note of names and acronyms that are used to describe equipment or parts; using these names will make the model familiar to all who know the process.
- *Draw or obtain a drawing of the system layout:* Before the tour, obtain a drawing of the facility, or while on the tour draw the facility and make notes on the drawing for each element in the system. After the tour is complete, this drawing will serve as a “trip to the floor” whenever you need to review the facility. Photos are also helpful, if allowed. Also, drawings can be used as backdrops for the simulation; that is, a drawing can be imported into the simulation software and the model developed to scale on the drawing image.
- *Get to know the system in detail:* One of the best ways to get to know the system is to spend time in the operations area and discuss the process with the operators—those that do the work. Remember, the more you know about the processes and capabilities of the system, the more capable you will be in modeling it correctly.

What is the objective?

Every simulation needs a defined objective.

Determining the objectives of a simulation project is a very important step since the simulation model will be designed around and focused on these objectives. It is important to realize that the objectives themselves may be the only understanding that management and others will have of the project. While the other steps will help manage the project, the objectives will manage the expectations.

The specific goals and objectives that are identified will be the criteria for success. The goals may be more general statements, but the objectives should include some specifics.

The following questions should be addressed and answered early in the simulation process as part of determining the project objectives:

- What is the primary reason for developing the simulation model?
- What is the value of performing the simulation analysis?
- What is the scope of the model?
- What questions will the simulation project answer; what problems will be solved?
- What issues need to be investigated?
- What are the appropriate performance measures (e.g., throughput, overall production time, resource utilization, system delays)?

What is the output?

The objectives indicate the performance metrics, but the format of the response should also be specified. This may include a mockup of how the final reports should look and what should be emphasized. The precise details will emerge during the project activity.

What is being simulated? (boundaries and assumptions)

A simulation model should contain only as much detail as is necessary to fulfill the objectives. Too much detail burdens the model building, slows the model run time, complicates the results and analysis, and inhibits effective reuse. More detail can always be added as it is needed. Other important parts of the functional specification are discussed below.

Simulation scope—specify the breadth and depth

The definition of the simulation scope is especially important as it defines the boundaries for the analysis. The boundaries should be kept as narrow as possible in order to allow the simulation to focus on a specific region. The following are example questions that can be posed in order to define a simulation's scope:

- Is only one section of a production line being simulated, or is the whole system, including raw material handling and warehouse, included?
- Is only the activity around an airport security station sufficient, or do aircraft arrival and departure details and gate information need to be included?
- Is only the workload of a bank teller important, or are the workloads of all bank departments to be considered?

Boundary Assumptions

Once the boundaries are defined, the assumptions of how the system being studied interacts with its external environment are defined. The following are some example questions that can be used to help define boundary assumptions:

- A production line being simulated may send finished material to a warehouse that is not being simulated. The entire warehouse doesn't have to be simulated, only its characteristics; for example, how often is the storage area full, or can it be assumed that it is never full?
- Will the needed raw materials for a production process always be available?

Operating Assumptions

Not every operation in the simulation scope may need to be simulated. There are some activities within the scope of the simulation that may be addressed in various

Identifying the simulation scope is the only way to keep the simulation project focused.

All simulation projects contain assumptions that should be identified early in the process.

ways. Consider the following questions to help you know what needs to be included in the simulation:

- Do all the steps need to be included or can some be combined? For example, the details of how an object is taken off a conveyor, manipulated to a particular position, and then put into a box may be combined into one operation. The simulation could take the object from the conveyor into a machine and then output the completed box.
- Are resources always available to do a task? What work breaks are to be included? Will supplies always be available? Will equipment ever break down or need to be resupplied or maintained?
- Will downtime be assumed from historical data or from a similar piece of equipment?
- What are the specific rules for transferring material from one location to another (will current practice or contemplated procedures be simulated)?
- What times are included with the processing time for a piece of material? For example, a fraction of the total setup time can be added to the overall operation time, thus eliminating the need for setup times to be defined for each part at each location. This proration of setup time is often done in various planning activities; however, this practice is discouraged in building simulation models. In the real system, setup is a separate activity from processing; therefore, it should be modeled that way in a simulation. The intermittent effect of setups in the real system is masked when using prorated values.

Specialized Logic

Specialized rules should be described that are either currently in place or planned. Consider the following examples:

- Dynamic changing of processing rates depending on operating conditions (e.g., speeding up or slowing down process times based on number of items in the system)
- Needing to call for a supervisor if particular events occur
- Unique rules governing the flow of materials from one source to another (e.g., tank levels, tank materials, length of time in a tank)
- Specialized cleaning and setup rules
- Constraints, if any, that are created as a result of the interaction with other system elements

Associated documents

The simulation will only be as good as the data that are used. Collecting that data can be very time consuming. It is important to develop and maintain a list of all the specific documents that are used in building the simulation model. Copies of the documents should be kept with the simulation project file or referenced for the simulation builders to use. Documents can include any of the following:

- CAD drawings, block diagrams, value stream maps, etc.
- Data tables or records containing the operation characteristics, such as operation times, move times, setup times, and downtimes, for each element in the system
- Lists of information and estimates developed for system elements when actual data is in short supply, or non-existent
- Spreadsheet files and any data that are imported to the model
- Detailed records obtained for each system element
- Operation times for each part and process
- How and when parts arrive at each location
- Production schedules
- Scheduled and unscheduled downtime, as well as information on time needed to make repairs and time between failures
- Changeover activity and setup time for different products or part types
- Material handling information such as speeds and capacity
- Any other information or data that is required for the system to operate correctly

Collecting data for use in the simulation is critical and can take more time than building the model.

Section 5-3 Prepare a flow diagram

The best way to document the material obtained for the functional specification is with a flow diagram of the system. Such a diagram also becomes a useful communication tool when discussing the project.

Most operating systems in manufacturing, service, logistics, or other areas can often be described at their simplest level with a diagram showing each step as a block. There are many ways to represent processes diagrammatically; for example, detailed CAD (Computer Aided Design) diagrams and P&IDs (Process and Instrument Diagram). The P&IDs provide very detailed descriptions (usually in 2 dimensions) of all equipment, piping, electrical, and other connections.

Another type of diagram, the value stream map, is often useful in modeling since it contains considerable process data that have already been investigated, discussed,

and compiled. An example value stream map is provided in Figure 5.3. This type of diagram, however, is not sufficient by itself for defining a simulation model since it typically lacks the detailed logic needed in most simulation models (e.g., rules for making material transfer decisions or detailed sequence of operations).

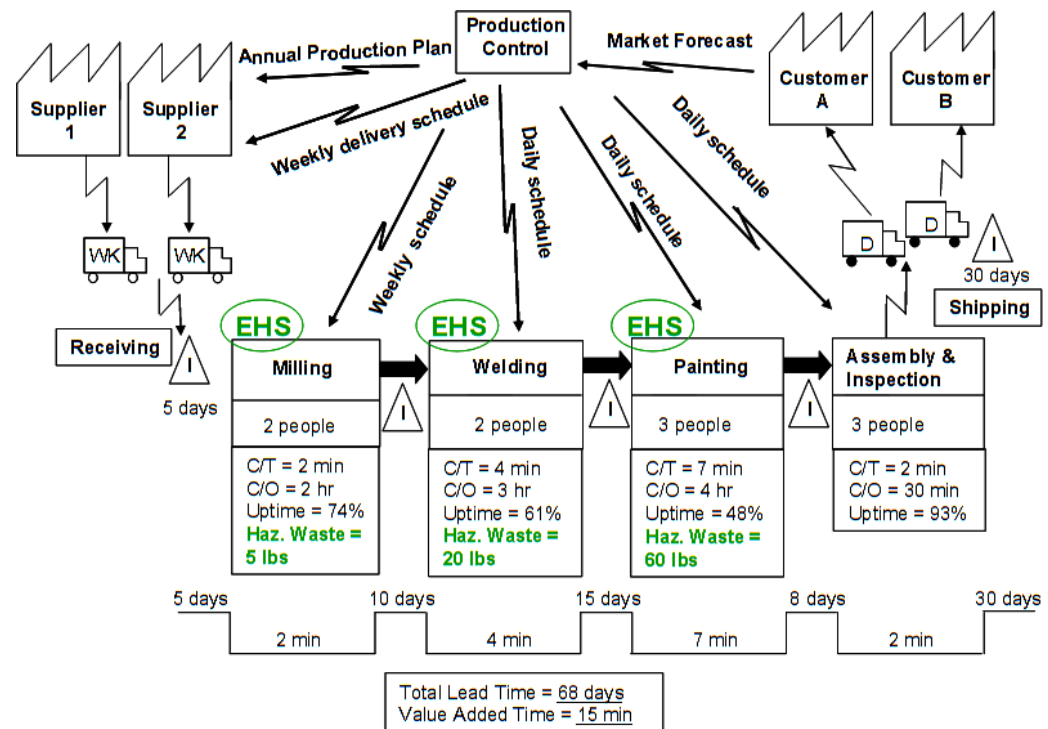


Figure 5.3 Value Stream Map as a starting point for simulations

The object flow diagram provides an opportunity for project team members to agree on how the system should be simulated.



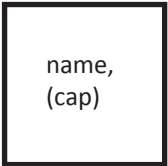
While there is no standard diagramming methodology, we suggest the use of an object flow diagram (OFD). It is specifically designed to support the conceptual simulation model design process; it does so by visually characterizing the key objects that operate in a system and by visually representing the relationships between those objects. Using an OFD forces the modeler to think through the many aspects of the system before translating mental models to a computer model—a step that is unfortunately all too often ignored.

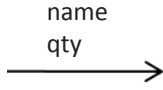

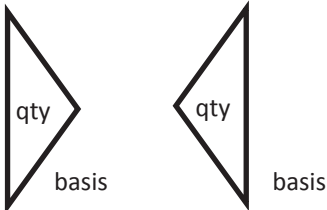


The OFD is the simulation modelers' "wireframe" or "schematic sketch" since it abstracts and outlines the basic structure of the system in simulation terms. The diagram helps the modeler recognize which objects and behaviors need to be considered, determine the level of detail needed to represent those objects, identify the types of data that may be needed, and communicate the process with non-modelers. On the other hand, the diagram is not intended to represent detailed procedures, complex logic, and the like. It is a high-level tool for thinking, communicating, and initializing model design.


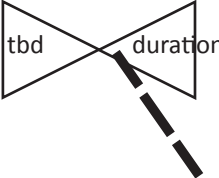

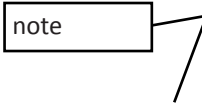

OFD symbols

Figure 5.4 provides a summary of the symbols used in a discrete OFD. The table provides a brief description of what the symbols represent and how they are used. The symbols are not specific to particular software but are designed to represent functions commonly used by simulators. Each symbol may or may not represent a specific object in any one simulator. For example, the function of bringing things into a simulation (source) may be a separate object in one simulator or included as part of a processing object in another.

In addition to the symbols, some information about the functions is included. This information provides additional detail about each function and helps to serve as a reminder to include that information in the simulation.

Figure 5.4 Object flow diagram symbols	
Discrete objects	Functional Description
Source (TBA, qty) 	Creates items that flow in the model; input system boundary TBA = inter-arrival time qty = quantity of items that enter the system at each arrival event
Sink 	Destroys items that flow through the model; output system boundary
Fixed Resource 	Generally remains physically in one place and takes action on items, such as processing, conveying, or storing. name = identifier cap = capacity of the object; maximum number of items that can be contained within the object. If capacity is not an issue, denote (inf) for infinite. Note: sides are significant. Use IDEF0s ICOM notation. <ul style="list-style-type: none"> • Left-side is for inputs to the resource, the primary item(s) that are transformed or consumed by the object. • Right-side is for outputs from the resource of the primary item(s) that is produced by the object. • Bottom-side is for “mechanisms,” other resources used to support operations of the object (e.g., operators, empty containers in which the items are placed). • Top-side is for “controls,” those things that constrain operations of the object or are conditions for functions of the object (e.g., messages, downtimes, object specific shifts).

<p>Flow/Route</p> 	<p>Link between fixed resources upon which items move, flow, or are routed</p> <p>name = identifier</p> <p>qty = quantity of items that flow together</p> <ul style="list-style-type: none"> Flows are assumed to require time for items to move between objects; however, the duration is not expressed in the diagram. Oftentimes the flow occurs with a mobile resource.
<p>Process</p> 	<p>Activity typically performed at a fixed resource and often-times in conjunction with a mobile resource. An activity may also be associated with a flow or move (e.g., a load or unload operation).</p> <p>duration = time to complete the activity</p> <p>There may be multiple processes conducted sequentially at a Fixed Resource, including a setup operation.</p>
<p>Group/Ungroup</p> 	<p>Combining items into a single unit or conversely splitting a unit into component items</p> <p>qty = number of items combined or number of items split</p> <p>basis = primarily used for combining, the criterion for grouping or ungrouping (e.g., combine based on product type)</p> <p>Typically these symbols are added to a fixed resource. If the symbol on the left is placed on the left or input side of the object, it denotes a combine operation. If the symbol on the right is placed on the right or output side of the object, it denotes a split operation.</p>
<p>Select</p> 	<p>Decision rule typically used by a fixed resource to either route an item out or “pull” an item into the resource.</p> <p>criterion = rule used to make the selection</p> <p>Typically these symbols are added to a fixed resource. If placed on the left or input side of the object, it denotes a pull operation; if on the right or output side of the object, it denotes a routing decision.</p>
<p>Mobile resource</p> 	<p>Generally transports items between fixed resources, carries out processes on items at fixed resources, or repairs fixed resources that are down.</p> <p>name = identifier</p> <p>qty = number of mobile resources needed to complete the operation</p>

<p>Communication</p> 	<p>Link between objects that involves information, messages, etc. These links do not represent the flow of an item.</p>
<p>Downtime</p> 	<p>Fixed resources and mobile resource may become unavailable for a variety of reasons (e.g., breakdowns, preventative maintenance, operator breaks, shift schedules).</p> <p>TBD = time between downtimes duration = how long the resource is down MTBF = mean time between failures MTTR = mean time to repair</p> <p>Objects may undergo several types of downtime; however, typically only one downtime symbol is used per object; details of the downtimes are included in associated text documents.</p>
<p>Performance measure</p> 	<p>System variable that is of particular interest. An output of a simulation model; sometimes referred to as a response variable or dependent variable.</p> <p>name = identifier</p> <p>This symbol is attached to the object of interest; for example, “average content” would be associated with a fixed resource; “utilization” could be associated with a fixed or mobile resource.</p>
<p>Note</p> 	<p>Annotation that provides information about the object</p>
<p>Questions</p> 	<p>Annotation on the diagram that raises questions that need to be addressed, identifies the need for additional information, or denotes issues that need to be discussed</p>

OFD example

Figure 5.5 shows a simple manufacturing operation, referred to as *simple cell*. In this case, items arrive by an overhead conveyor, are placed onto a workstation by an operator, and are moved to a pallet once a finishing operation is performed. Notice the items are gray when they arrive on the overhead conveyor and are blue when placed on the pallet. When a pallet is full, it is picked up by a fork truck.

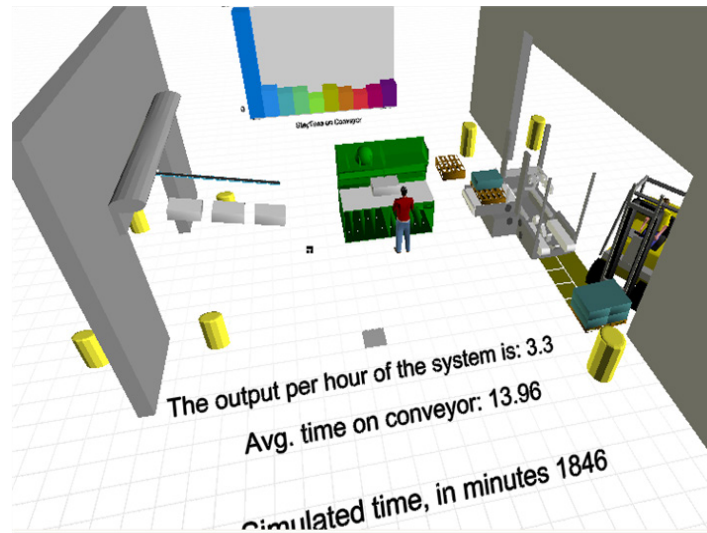


Figure 5.5 Simple cell, manufacturing operation

From a modeling perspective, what do you see in the picture? What are the key elements, and their relationships? Before reading on, sketch out an OFD based on what you see in the picture and using the symbols in Figure 5.4. There is no need to draw it on the computer, just sketch it on paper. In fact, in practice, our initial sketches of the system are always done by hand. The diagrams will change many times as you learn more about the system and understand its behavior. Of course, for presentations and a final report, you should neatly draw the diagram using a computerized drawing program (e.g., MS PowerPoint or Visio).

After sketching the system in OFD format, compare your diagram to the one in Figure 5.6.

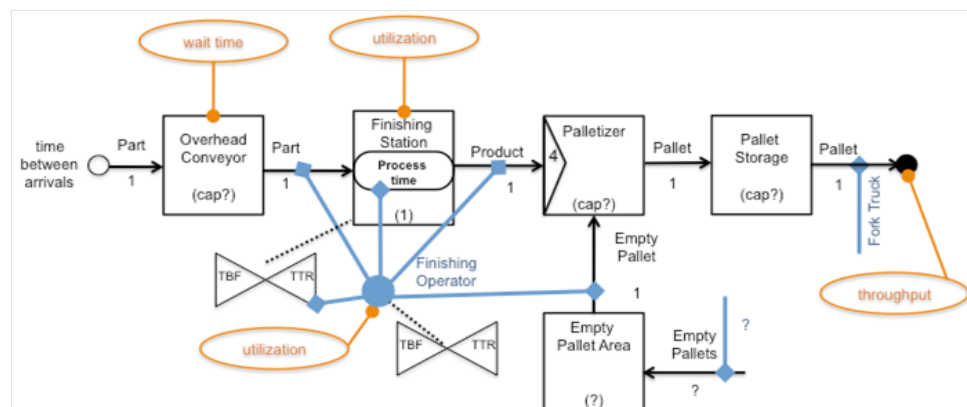


Figure 5.6 Object flow diagram for Simple Cell

Consider the following description of the diagram. The boxed-shaped objects in the OFD represent the fixed resources, such as the overhead conveyor, workstation,

palletizer, storage areas for the items on pallets, and empty pallets. Items (e.g., parts, products, and pallets) flow between these resources. The finishing operator moves the items from the conveyor to the workstation and then on to the palletizer. The operator also performs an operation on the item at the workstation, places empty pallets on the palletizer after a full pallet is moved to a storage area (where it waits to be picked up by a fork truck), and performs repair operations on the station when it goes down.

The bowtie-shaped symbol indicates the downtimes that need to be considered on the resources. In this case both the finishing station and the finishing operator have downtimes. The finishing station likely requires some scheduled and/or unscheduled maintenance, and the operator takes breaks and therefore is not available to perform work. The diagram also denotes the key performance measures of interest. Based on the picture, it appears throughput and the time an item spends on the conveyor are of concern; on the OFD, we also include utilization of the operator and workstation as performance measures of interest.

Section 5-4 Initial sizing of resources

In the example above it was clear how many resources are in the system; however, this will not be the case for all systems. For example, the system may not yet exist, and the simulation model is being developed to analyze and assess the proposed system. Oftentimes, the model may be used to determine how many transporters are required; (e.g. car carriers in an automotive assembly plant). It would be a waste of time to keep incrementing the number of carriers and running the model, guess-and-check fashion, to reach a feasible solution; therefore, this section provides a methodology to determine the minimum number of resources required.

The approach is based on the notion from queuing theory that the number of resources available must be able to keep up with demand; that is, the overall service rate, μ , must be greater than the arrival rate, λ :

$$\frac{\lambda}{\mu} < 1.$$

Stated another way, and shown in the expression below, the ratio of the total amount of work that needs to be performed must be less than the capacity of the resources; therefore, the minimum number of units of the resource is the next highest integer of the ratio:

$$\frac{\bar{N}_T * \bar{W} + O_T}{n * R_T * \left(1 - \frac{\bar{DT}}{R_T}\right)} < 1$$

$$\therefore n = \left\lceil \frac{\bar{N}_T * \bar{W} + O_T}{R_T * \left(1 - \frac{\bar{DT}}{R_T}\right)} \right\rceil$$

where

n = min. number of units of resource;

\overline{N}_T = mean number of items arriving at the resource in time period T ;

\overline{W} = mean amount of work to be performed per item;

O_T = amount of other, non-item work performed in time T ;

R_T = amount of time resource is available in time T ;

\overline{DT} = mean downtime in R_T .

Consider the following example. Swell Computers is considering starting a new facility to produce its newest model. They expect orders to arrive at their website 24/7 at an average rate of 1.5 orders per hour. They estimate the assembly operation to take 35 minutes, on the average. The operators are required to clean up their workspace and compile their paperwork at the end of the shift—this typically takes 20 minutes. Assembly will only work one shift or 40 hours per week; operators take 2-15 minute breaks and a 30-minute lunch per shift. Swell wants to estimate the number of assembly resources it will need.

The sizing expression is applied below and indicates at least 5 assembly resources are needed. Of course, a simulation model would consider the effect of variability in the order arrival times and assembly times, as well as interactions with other production processes, level of service, etc. Nonetheless, the simulation model would be built with 5 assemblers as a starting point, as shown below:

$$\left[\frac{(168 / \text{week} * 1.5 \text{ orders} / \text{hr}) * 35 \text{ min.} / \text{order} + 20 \text{ min} / \text{shift} * 5 \text{ shifts} / \text{week}}{(40 \text{ hr} / \text{week} * 60 \text{ min.} / \text{hr}) * (1 - \frac{1}{8})} \right]$$

$$= \left[\frac{8920 \text{ min. of work}}{2100 \text{ avail. min. per assembler}} \right] = [4.2] = 5$$

Section 5-5 Building the simulation

The second phase of the project concerns the physical work of the project team. This part of the documentation template details how the simulation model was built and how it operates. This information is key for present and future users who want to successfully make use of, and modify, the simulation model.

While the first section of the template should be stable; meaning, the objectives should remain consistent throughout the project, this section is a “living” document and will change and mature as the project proceeds. It should be updated during the project so that any new member who may join the team will have an initial reference point.

The section starts by documenting the simulation software used, as well as the software version number. Other items to be documented include details and defini-

tions of the basic parameters used, such as the base unit of time and distance; abbreviations and acronyms; and any color coding standards. Additionally, any simplifications that are being made for the simulation should be listed; such simplification may include combining a machine's setup time with the processing time or combining all breakdowns into a single common event.

The documentation should include all parameters associated with each of the simulation objects as well as any specific logic used in their operation. There should be a description of what data is used and where it is stored (e.g., in tables or on individual objects). One of the most important parts of building a simulation model is to fully define and document the use of variables, attributes, and functions. If possible, this definition should be done within the simulation software. Much of a simulation model is self-documenting as a result of entering the data required to build the model; however, the more complex the logic within a model, the more important it is for information about variables and functions to be documented.

Using comment lines within the code segments to explain logic wherever possible makes it easier for others to understand. Logic that may seem obvious at the time it is written may not be so obvious to another person in the future.

Areas where specialized logic is developed should also be explained in Part II of the template documentation. That documentation should include why the custom programming is needed and a detailed description of how it is implemented.

Additionally, if any new or customized objects are developed for the simulation, they should be documented in the template. A description of the object, how it works, any limitations, and the possibility for re-use should be included.

Any use of subroutines or specialized program files should always be included in the model documentation. It is important to include everything that is used to make the model run.

User interface

Since the simulation may be run by individuals who were not involved with its development, a description is needed of how a user interacts with the simulation and where the results can be found. Screen shots of the simulation will help further define the model's operation.

For data input, the documentation should identify the key variables and describe the organization of the input data sheets or tables. It should describe how and where data may be changed.

All reports or other outputs of the simulation project should be documented with samples. Details of how the reports were calculated should also be noted. Instructions for exporting, including any templates that were established for the project, should be included.

Section 5-6 Results and analysis

The third section of the project template documents the completion of the project. It has two major functions. First, it documents that the simulation is operating correctly. A validation plan describes how the simulation is checked. This often is done by checking output for known conditions or checking output when all randomness is removed. Second, it documents the procedure for defining how the simulation is used and how the results are analyzed.

Validation

Model validation has two parts. First is an operational check that the input data and input methods are correct, as well as the way the output is calculated and presented. For example, if someone created a program to add numbers you would try entering a known quantity such as a 1 and a 2 to see if the answer was 3. Second is a validation that the model has successfully captured the operational characteristics of the system.

A simulation model that looks like the real system, oftentimes obtained through 3D graphic capabilities in the simulation software, can facilitate validation, enhance credibility, and increase confidence in the simulation. Of course, the look of the model cannot supplant a correct representation of the system, appropriate modeling logic, correct input data, and valid statistical analysis of the model's output.

Operability

The following list contains criteria that must be met for project operability:

- The data is checked by personnel who understand the operations to ensure that the data still correctly represents the system
- The reports are accepted as being correctly generated according to calculated values similar to the actual production reports. The data entry methods are understood by all involved in the model building.

Validation

The following list contains criteria that must be met for project validation:

- The operation of the model should be checked against the actual system that is being modeled by using variable values that mirror current operations. The results should reflect the general operating characteristics of the actual system and people who are familiar with the operations should agree that the simulation correctly captures the system characteristics.
- If the system is new, the simulation should be checked against the functional specification and be reviewed by developers and operations personnel.

A simulation should “look” and “feel” like the actual operation to those involved in the project.

- The basic rule of thumb for model validation is that the model behaves as expected. If it doesn't, check the model logic and make sure it is working correctly.

The following are specific areas that should be checked, if applicable to the project:

- Material balance (often checked by turning statistical distributions off)
- Downtime/reliability (note number of downs, total downtime, etc.)
- Production schedules (run variation of schedules to assure that changes are handled correctly)
- Specialized logic (test individual units and step through operations if necessary)
- Visual characteristics (Does it “look” correct? Would an operator say “that’s the way it works”?)

A basic fact is that a simulation will never act exactly like the real life process as there are just too many variables, but it should capture the major operating characteristics of the system. A rule of thumb is that the model behaves as expected. Success can come when someone familiar with the operation can look at the model and results and say “that’s the way it works.”

Analysis

Once validated, the model can be used for extensive analysis. Simulation analysis is a process itself (see Chapter 10). It requires a plan to ensure that the analysis will be consistent with the project’s goals and objectives.

The focus on the output analysis should be on the performance measures defined in the functional specification. Simulation software can often be intimidating and require a user to “cut through the smoke” and find the important results. Correctly defined performance measures should help identify what’s important and what’s not. A major part of the analysis plan is a well thought out design of experiments.

The design of experiments defines how the simulation is to be run. It does so by specifying such things as what variables should be changed, how many runs should be conducted, and what results should be collected to provide the information needed to achieve the goals and objectives of the simulation project. Experiment design specifies different simulation scenarios, which in turn define how the system configuration will vary. Results from the experimental runs need to be statistically analyzed. The definitions at this point may include what specific statistics should be collected and the level of confidence desired.

Based on the objectives, simulation experiments are defined to obtain the required output from the model.

- *Define the base scenario:* The base scenario is the basic system configuration; it is typically a representation of the current system. This scenario is used to validate the model and all other scenarios are compared to it.
- *Define the changes that need to be investigated:* The changes that need to be made will correspond directly to the changes defined in the goals and objectives. For example, machines may be added, operation times changed, or different methods of material handling used.
- *Establish the number of scenarios for each change:* There needs to be a limit on what is changed for each scenario. If too many things are changed, then it will be impossible to determine what change caused what effect.

The simulation output is the final step of the simulation project process. It needs to be stressed and understood that if any of the previous steps have been neglected or performed incorrectly, then the results may be invalid. Good output analysis involves correct application of statistical analysis.

The focus on the output analysis should be on the performance measures defined in the functional specification. Simulation software can generate a lot of output by default. The developers try to make as much information available to the user as possible. The volume can be intimidating and requires a user to be able to distinguish the results that are relevant and important to the project at hand. Correctly defined performance measures should help identify what's important and what's not.

With the important data identified, make charts and graphs that will visually convey this information to management. The way the data is presented is sometimes more important than the model itself.

The full template is contained in the Appendix.

Section 5-7 Example of a project template

This section provides an example of how an Occasional User might utilize the template to define a simulation project for an aluminum plate production line. The following includes background information on the problem. Part I of the template has been completed based on the background information.

Allied Aluminum Company—Dexter Plant

Polished Aluminum Plate Production Line

Background

Aluminum, along with other construction materials, has experienced a downturn in business. New uses for aluminum in ship building and other areas, has caught the interest of Allied Aluminum senior management. The marketing organization has indicated that to be successful, the company will have to be able to react to orders

of varying sizes of aluminum plates. Allied Aluminum has two plants capable of producing the aluminum plates but neither plant is performing well.

Four years ago the Dexter plant put in a new high speed cutter line in an attempt to increase production and be more efficient; however, in comparison to other plants, the plant still lags in linear footage across its new cutter and in equipment utilization..

The Dexter team has come up with a number of possible areas where they can improve performance. They are focusing on adding a new conveyor that will eliminate delays caused by the manual transfer of material in the plant. The union has also agreed to discuss work rules and downtimes. To get the work done as quickly as possible, the team agreed to outsource the simulation development but wanted to be involved with the analysis. The development will be based on a functional specification using the simulation project template.

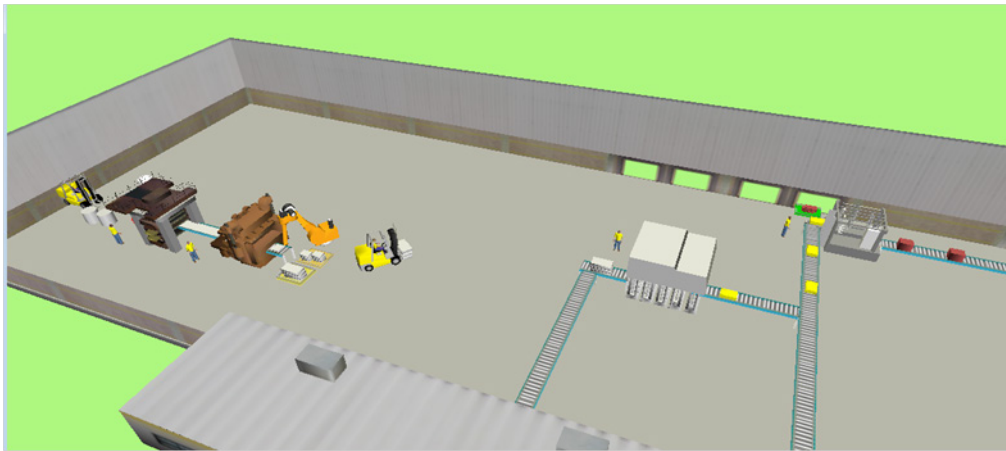


Figure 5.7 Dexter high speed line overview

Plant layout

There are two production areas in the plant—a newer high speed line and an older, conventional cutting line. Both lines utilize the same wrapping and packaging system as shown in the figures.



Figure 5.8 Dexter plant cutting and shearing

Aluminum coils of varying lengths are brought to the high speed cutter (HSC) where they are loaded on to the cutter and cut to the correct width. Protective paper is matched with the aluminum at this time. This paper serves as a protection for the polished surface of the plates during the stacking, packaging, and shipping of the product. The cutter produces the correct width while the shear section cuts the length. There are two stacking sections at the end of the HSC. The stacking sections are used alternately when possible. Regular orders move on to be wrapped while bulk orders go directly to the warehouse.

Stacks currently are moved by fork lift to the warehouse or automated wrapping unit (AWU) input conveyor. Units are wrapped with foil. Wrapped stacks then either move on to the automated packing unit (APU) or go to a unit pack station by conveyor. The AWU gets packs to wrap from the HSC or packs produced from the older, conventional cutter line.



Figure 5.9 Dexter plant wrapping and packing

Finished production is moved to warehouse areas by fork truck.

The first section of the project template might look like this when filled out:

Dexter Plant Productivity Project Worksheet

Date Modified: May 17, 2011

Simulation Name: HSC cutting line

Key Words: Continuous Improvement, High Speed Cutting, Production Scheduling, Work Rules

Part I

Functional Specification

General Description

While the new high speed cutting line has increased the plant's ability to produce plates to order, the plant still lags behind other facilities in overall throughput and efficiency. Unless changes are made, new production orders will be issued to other

facilities resulting in loss of revenue and jobs to the Dexter site. Simulation will be used as the primary analysis tool to help the plant's continuous improvement teams validate and prioritize their recommendations.

Goals and objectives

- Create a simulation that can be used by the continuous improvement (CI) team members
- Validate the simulation
- Use the simulation to determine the most cost-effective actions

Outputs/metrics

- Linear footage across the high speed cutter
- Number of packs delivered to the warehouse per shift
- Machine efficiency for the cutter, shearer, wrapping, and packing units
- Percent orders delivered on schedule

Simulation scope

- All operations within the new cutter line building
- All operations requiring staff

Boundary assumptions

- Steel and paper rolls are always available and brought in to keep floor areas supplied.
- Corrugated box material and foil wrapping is always available.
- There are no bottlenecks moving material out of the area.

Operating assumptions

- A product list of plates produced on the line is available.
- Specification for coils of materials is available from the purchasing department.
- Current production schedules are available from the planning department.
- Future schedules will be provided by corporate marketing.
- All operating rules, machines rates, and control logic are included in the Operating Procedure book.

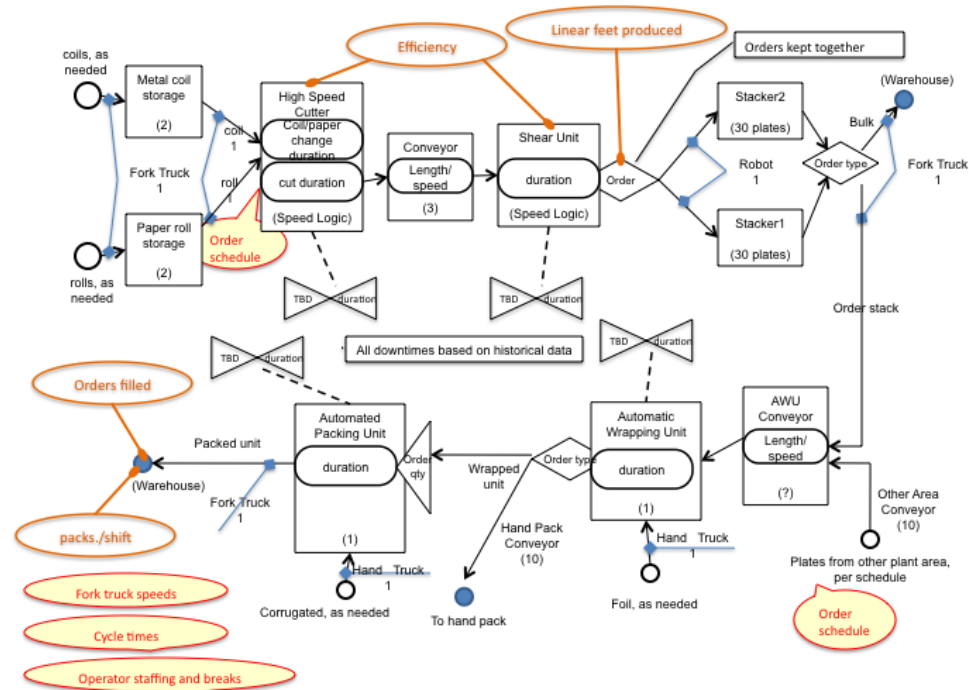


Figure 5.10 Object Flow Diagram

- Exact layout information and conveyor speeds are contained in plant CAD drawings.
- Historical machine downtimes are recorded in the control system and will be used as a starting point. Impact of improving maintenance, response, and repair times will be determined by the simulation.
- Current work rules governing break times and staffing levels will be used but may be modified in the simulation.

Specialized logic to be included

- Break times and schedule maintenance times per schedule
- Speed through the cutter system change dynamically based on
 - length into coil;
 - inspections;
 - metal and paper coil changes.

Exercise 5-1 Fister's Frozen Foods

This is the first of many exercises in this book. Each exercise will help to develop simulation skills. All the exercises follow the same pattern and use simulation to address operational issues or problems. Most of the examples have been taken from actual experiences; however, to maintain anonymity the names have been changed. Each exercise includes the following sections:

- *Background:* Provides information about the general nature of the company and the situation that has brought about the particular problem or issue; also indicates the relationship between the person using simulation as a tool and the owners of the problem
- *Problem statement:* Poses a question about the operations that needs to be addressed
- *Expected results:* States what form the response to the problem statement should take (e.g., a report, analysis, presentation, etc.); the audience expecting the results may also be provided
- *Operating data or additional information:* Contains specific information about the system in question; data may be from historical information or previous analysis; as in real life, may provide more information than is needed for successful completion of the exercise
- *Notes:* Provides information on procedures or concepts needed for the exercise; there are always many ways to solve a problem and these steps show one possible way

Background

Arnold Fister's family is in the business of quick-freezing gourmet food that is sold through the internet. Over the years their operation has grown from their garage to a new plant. Once frozen, it's important to get the food into shipping containers and into the mail as fast as possible. In the main section of the plant, which is kept at a low temperature, the food orders are assembled and frozen. The frozen pieces are placed in boxes and sent to the shipping area.

The boxes are moved by a robot from the cold area of the plant to the shipping area, which is not refrigerated. In the shipping area, the boxes are taken from a conveyor to the packing machine where the proper amount of dry ice is added and the box hermetically sealed. Shipments are divided into two categories depending on the destination distance and region. Those travelling the greatest distance or going to southern locations require more dry ice in the boxes.

After being sealed, the boxes are placed on pallets by destination type. Once four boxes are placed on the pallet, the pallet is moved by overhead crane to a pickup location where a fork truck takes them to waiting delivery trucks.

A single operator has been able to manage the shipping area. The operator must pick up the box from the conveyor and bring it to the packing machine and set the

All the exercises in this book follow the same format.

machine for the proper amount of dry ice. The operator also takes the completed package from the machine to the appropriate pallet. Once a pallet is moved, the operator places a new pallet in place.

Problem

Business is picking up and there is a worry that the operator will no longer be able to keep up with the packaging process. It's critical that boxes not stay in the shipping area for long periods before being packed for shipment. Options include adding more staff, adding automated loading and unloading for the packing machine, or adding bar coding so that the packing machine does not have to be setup manually by the operator.

Expected Results

Fill out Section I of the template, including the drawing of an object flow diagram. Decide on the objectives for the project, the scope, what information should be included, and what assumptions are valid. Also decide what scenarios should be included in the simulation analysis.

Additional Information

Boxes needing more dry ice are color-coded red while standard boxes are coded blue. Some important measures would include the percent of the time the operator is busy and the length of time a box stays in line before packing. For the project, consider what other information is needed and indicate that on the project template.

Figures 5.11, 5.12, and 5.13 give an indication of the plant layout.

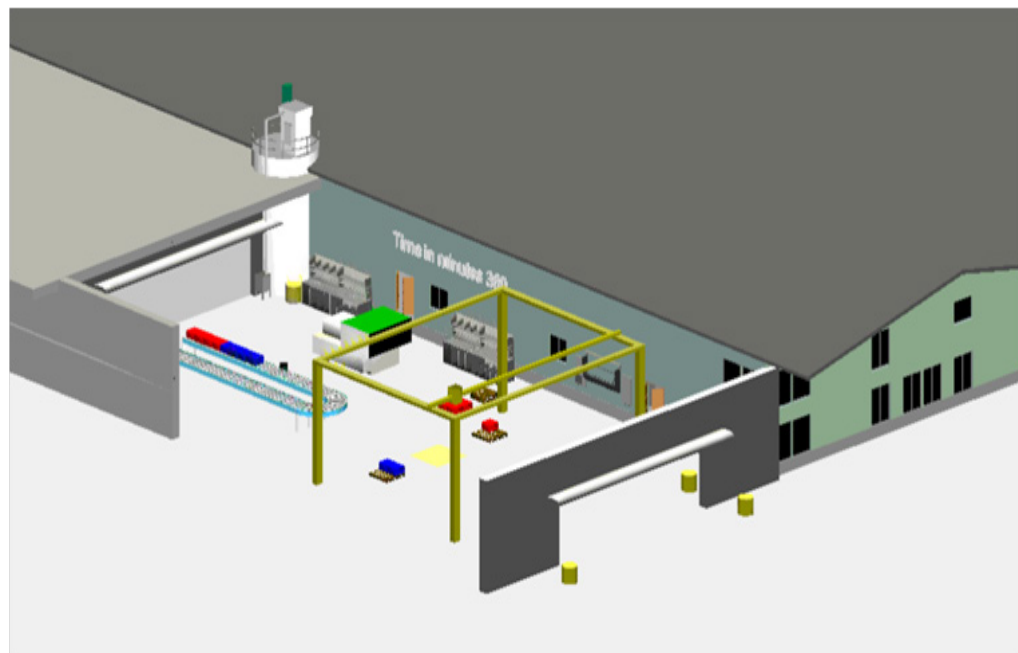


Figure 5.11 Fister's Frozen Food plant and shipping area

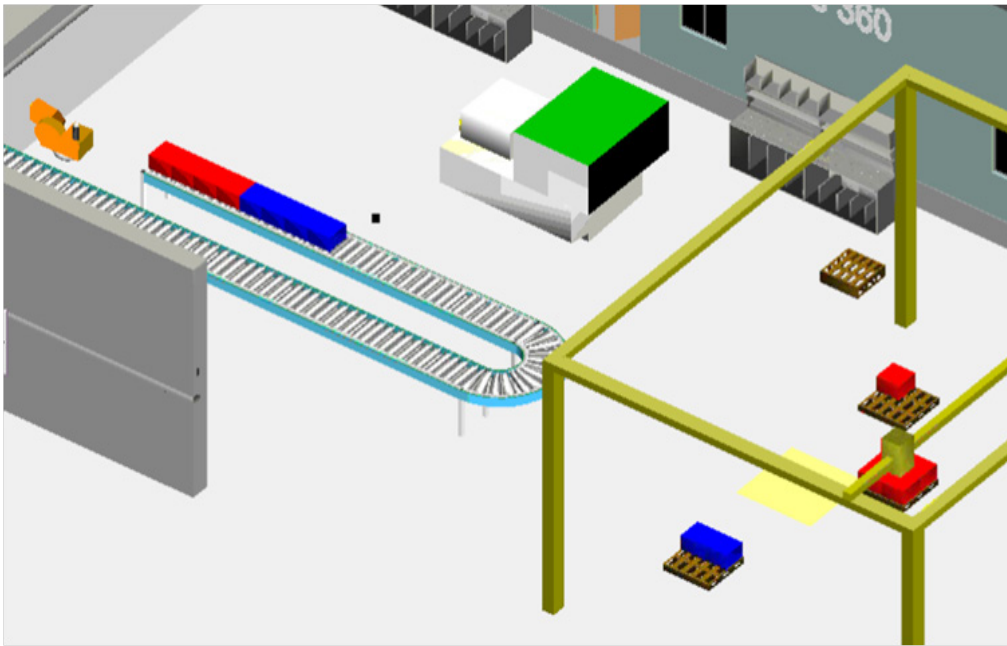


Figure 5.12 Robot moves boxes from frozen plant to shipping area

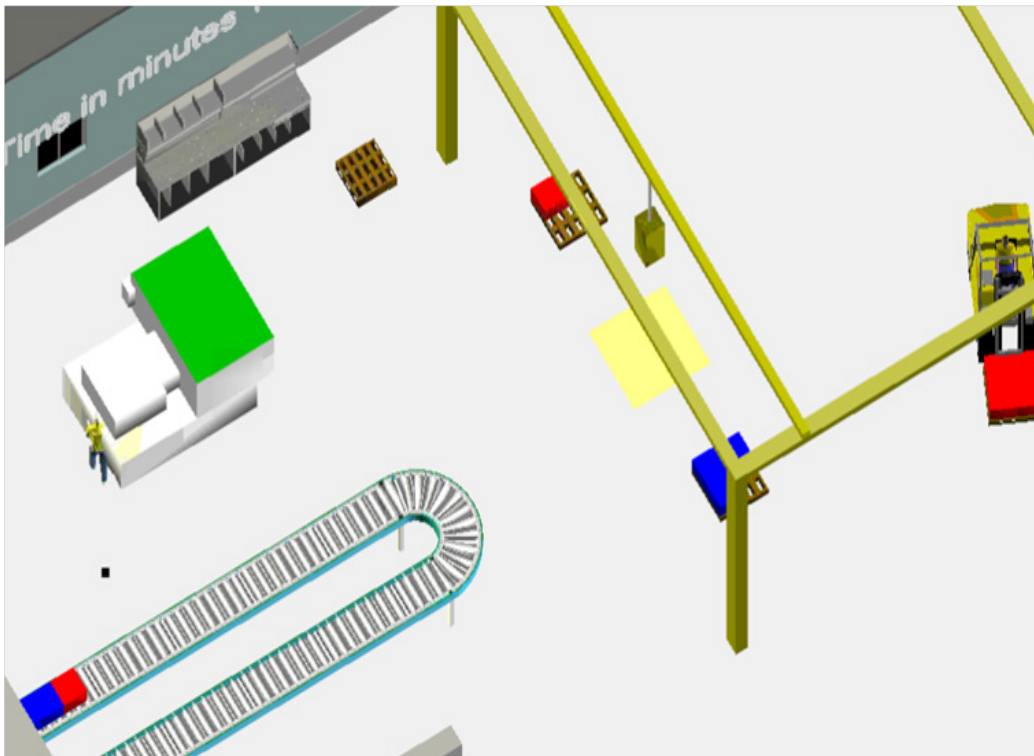


Figure 5.13 Operator moves boxes from conveyor to packing machine then on to pallets. Crane moves pallets to pickup area

Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Describe the main activities that need to be performed in the four key steps in a simulation project.
3. Discuss why it is so important to have a clear objective for the simulation project.
4. Discuss the difference between boundary and operating assumptions. Provide examples of each.
5. Describe the significance of each side of the fixed resource symbol that is used in an OFD; in other words, what is inferred from what flows in and out of each side.
6. Using the OFD notation, represent a technician (a mobile resource) that takes a 15-minute break every 2 hours.
7. Using the OFD notation, represent a 100-foot conveyor that travels at 300 feet per minute and breaks down periodically. The time between breakdowns is exponentially distributed with a mean of 4 hours; the time to repair the conveyor is exponentially distributed with a mean of 120 seconds. Items traveling on the conveyor are 2-foot cubes.
8. Compu-Help is considering a new call-in help system. They are planning on using automated help desk devices that operate 24/7. They expect calls to arrive at an average rate of 15 per hour, Poisson distributed. Each device can handle one call at a time. The time spent with each caller is normally distributed with the mean time of 3 minutes and a standard deviation of 1 minute. The system is designed such that a maximum of 6 calls can be waiting at any time; i.e., if a call arrives and there are six waiting, then the incoming call is lost. The call system is expected to crash occasionally—the time between crashes of each device is exponentially distributed with a mean of 4 hours; the time to recover is uniformly distributed between 4 and 8 minutes. Construct the object flow diagram (OFD) for this system.
9. For the Compu-Help problem, calculate the minimum number devices needed to meet the anticipated demand.
10. For the Compu-Help problem, assume the time to handle each call is triangularly distributed with the minimum time being 1 minute, the maximum time 20 minutes, and the most likely time 9 minutes. Also, assume each device must process a report on its activities every 4 hours; the time to process the report is uniformly distributed between 4 and 6 minutes. Calculate the minimum number devices needed to meet the anticipated demand.

11. Describe various approaches that are used to help establish the validity of a simulation model.
12. Describe the process for designing experiments in a simulation project.
13. Discuss the role of performance measures in analyzing simulation output.

The Intermediate User

Chapter

6

Building Basic Simulation Models

The Intermediate User uses simulation as part of his or her job description, but it is not usually the main focus. Typical job titles of the Intermediate User include operations analyst or industrial, process, packaging, or manufacturing engineer. Individuals engaged in manufacturing research and development may also be Intermediate Users. Typically, the Intermediate User will be involved with simulation five or six times per year. Intermediate Users can build relatively straightforward simulation models using drop-down menus or wizards. When more detailed simulations are required, the Intermediate User can call on technical support or third-party consultants.

This chapter introduces the basic structure and components found in most simulation software modeling and analysis packages. It includes a discussion of the modeling environment, introduces the basic structure and functionality of modeling objects, defines relationships among objects, describes how to move items through a model, and discusses obtaining output statistics from a single run of a model. This chapter will illustrate the object functionality as used in the *FlexSim* application. The remainder of the book builds the reader's capability to create and analyze simulation models through the *FlexSim* software. While *FlexSim* is used in this explanation, the functionality is similar to other applications, but is implemented differently.

This chapter, and those that follow, contains exercises that will develop those simulation skills used by Intermediate and Advanced Users. These skills, which are needed to build simulations, are discussed in general terms in the main chapters, while specific details of the *FlexSim* implementation are contained in the Appendix.

The discussions in this chapter are meant to be a quick-start introduction to simulation modeling. Some of the topics will be explored in greater detail in later chapters. Consult the Appendix as well as the Users' Manual in the Help section of *FlexSim* for further information about using the software.

Using simulation as a basic tool can be accomplished with the use of pre-built logic and a minimal amount of training.

Section 6-1 Simulation environment

FlexSim, like other professional simulation software, provides extensive support for building and analyzing simulation models. Figure 6.1 outlines the basic components found in such software. Most of the components, especially those in the left portion of the figure, will be discussed later in the book.

This chapter concerns the modeling environment and focuses on the components in the right portion of Figure 6.1. All simulation software provides a set of pre-defined modeling objects that facilitates model building. Such objects include queues that store items awaiting processing (due to unplanned delays), processing objects that modify or create planned delays for items as they flow through the model (service operations), transporting objects that move items through the model, etc. The number and capability of such objects varies widely in the various simulation software products that are on the market today. Some software, like *FlexSim*, allows users to easily change the behavior of objects, and Advanced Users can create their own objects.

The simulation environment for most software applications is similar and follow generally accepted conventions.

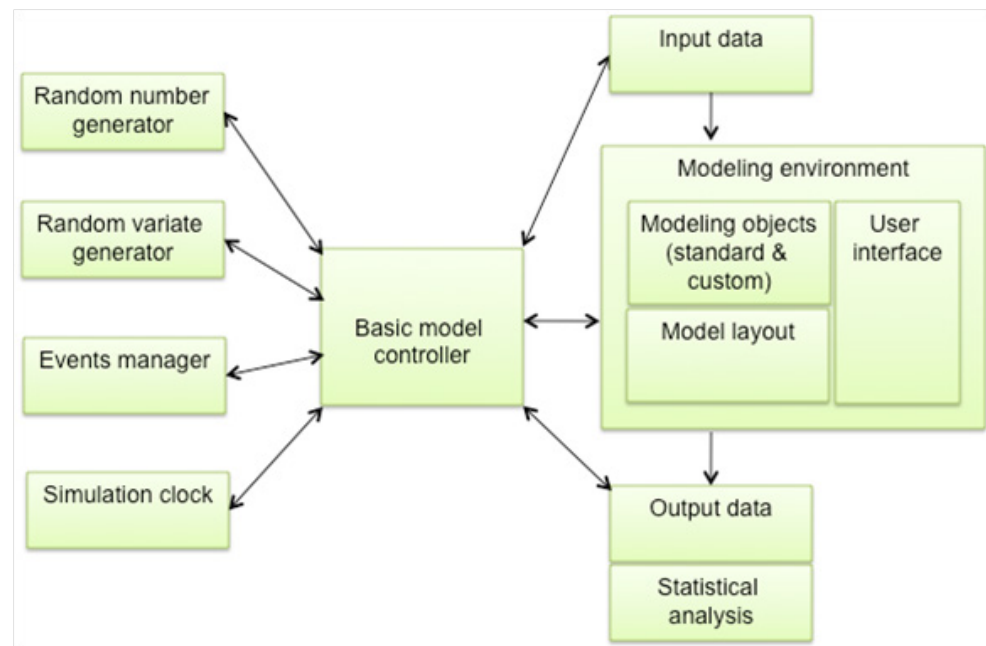


Figure 6.1 Basic components of simulation software

The modeling objects are typically placed and arranged on a layout, or simulation surface, that resembles the physical layout of the system being modeled. The objects are connected to represent the flow of items through the model and permit communication among objects.

All simulation software products have their own user interface that enables users to effectively and efficiently build models, import data, analyze results, etc. While much of the underlying logic is the same across the various simulation software products, the user interface and accessibility to the underlying logic varies extensively.

FlexSim's interface is quite effective for all levels of users; however, much of its power lies in its openness and ability to modify the objects and their behavior to best match the modeling requirements.

As mentioned above and shown in Figure 6.1, a simulation model is data driven; that is, without data, the model is useless and cannot function. Simulation software, therefore, needs to provide mechanisms to easily import data. Similarly, the reason we build and execute models is to obtain information and to understand the consequences of actions; therefore, simulation software needs to provide mechanisms that easily present or export data and that facilitate the analysis of that data.

Simulation environment terminology

When describing a simulation, certain terms are used to talk about the component pieces. While the specific names may vary in each software application, the concepts and functionality are basically the same.

Time and space

Time and space in the simulation environment are dimensionless until the user decides what the dimensions should be. When a new model is opened in *FlexSim* a screen appears, as shown in Figure 6.2, showing the default definition of units. The values are used during some reporting and internal calculations. All data input for the model should be consistent with these units. For example, if the time unit is seconds, the time between arrivals and cycle time for a processor would both have to be expressed in seconds. As a result, statistics that are generated by the simulation, such as the amount of time a processor is operating, will be interpreted as being in seconds.

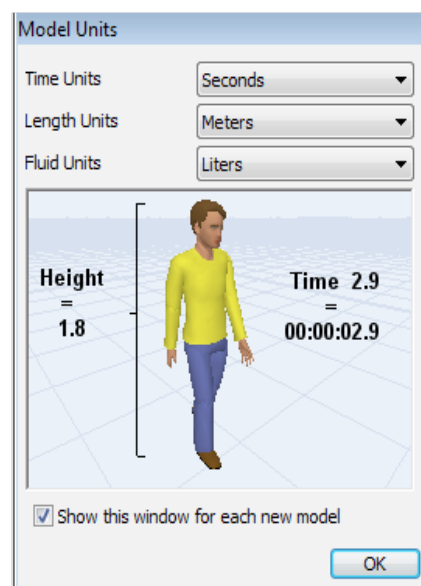


Figure 6.2 Units screen

Space is also dimensionless. The default unit of space on the simulation layout screen is one grid unit of size 1x1x1. The user has to decide the actual units that the default grid space represents. Equipment can be scaled appropriately. Task executors, such as fork trucks, have speed values assigned to them as the number of grid units traveled per clock unit. Being consistent with data values is critical to a successful simulation.

The main screen

Chapter 3 dealt with models that were already built along with the simulation environment to run them. This section starts with a blank page or layout known as the main screen as shown in Figure 6.3.

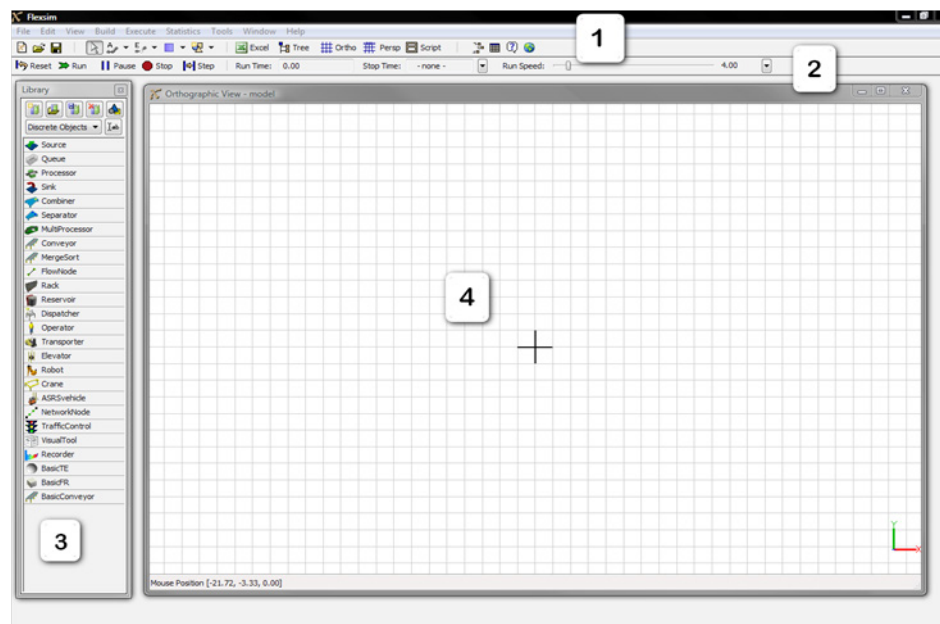


Figure 6.3 *FlexSim* model development main screen interface

As discussed in the previous section, all simulation applications have an environment in which a simulation model is built. That environment contains the surface for building the model (sometimes referred to as a layout), menus for controlling files and functions, and basic units or objects that are used to build the simulation model. *FlexSim*, as with most other simulation software, uses the conventions associated with Microsoft applications (file management, drop-down menus, tool bars, etc). As a side note, in the student version of *FlexSim*, the watermark “Educational Version” appears on the layout.

The main screen is made up of four major areas:

Area 1: Main toolbars

- *Top row*: Set of Microsoft *Windows* style drop-down menu options for file control as well as *FlexSim* operating controls
- *Lower row*: Shortcut buttons that allow quick access to some common *FlexSim* interfaces, operating elements, and model views

Area 2: Simulation control panel

- Buttons
 - *Reset*: Initializes the model
 - *Run*: Begins execution of the simulation
 - *Stop*: Stops the model at the end of the current clock cycle (A simulation can be re-started from the stop point.)
 - *Step*: Moves simulation ahead to the next scheduled model event
- Time controls
 - *Run Time*: Displays the current model time in simulation time units
 - *Stop Time*: Sets a specific time at which the simulation will be stopped (Once stopped, the value can be changed to set a new future stop time)
 - *Speed Slider*: Defines the number of simulation time units per second of real time

Area 3: Library icon grid

- Objects that can be brought into the simulation model by clicking and holding down the mouse button while dragging the object onto the modeling surface or layout

Area 4: Model view window

- Surface where the simulation model is built

Most applications allow modifications to the basic screen. The Appendix shows how the *FlexSim* simulation workspace can be visually changed.

Section 6-2 Simulation components

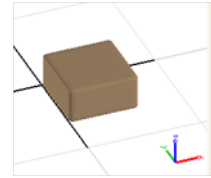
Graphically-oriented simulation software applications make use of elements that are placed on the simulation surface and can be viewed. In *FlexSim*, an *object* is the most basic building element of a simulation.

In *FlexSim* there are two basic types of objects: *discrete* and *fluid* (continuous). The discrete objects are used to develop discrete-event simulation models where model behavior results from events that occur at discrete points in time, such as an item arriving to the system or a machine stopping due to an internal failure. Fluid or continuous objects are used to model behavior that results from changes that occur continuously over time, such as the filling of a tank with a liquid. Most of the

Simulation objects perform specific functions while flowitems move within the simulation.

focus of this book is on discrete-event models and the use of discrete objects. Fluid objects are discussed in Chapter 14.

Simulations normally involve actual discrete entities that physically move around in the simulated environment. In *FlexSim*, these entities are called *flowitems*. Depending on the simulation, these could be boxes, products, customers, paperwork, and so on. Without flowitems, there isn't a need for simulation. Some simulation software packages refer to flowitems as entities or transactions.



In *FlexSim*, flowitems are listed in the Flowitem Bin which is accessed through the Tools menu at the top of the screen. Normally, flowitems are brought into a simulation through the source object where the choice of using a 3D flowitem is listed in a drop-down menu. Additional details are included in the Appendix.

Simulations also need objects that interact with flowitems. These might perform an operation, create a delay, or move the items. In *FlexSim*, there are two general categories of such discrete objects: fixed resources and task executors (mobile resources).

A *simulation model* is simply a collection of these objects put together in such a manner as to simulate the behavior of a system. The *FlexSim* simulation model shown in Figure 6.4 is one such combination of objects.

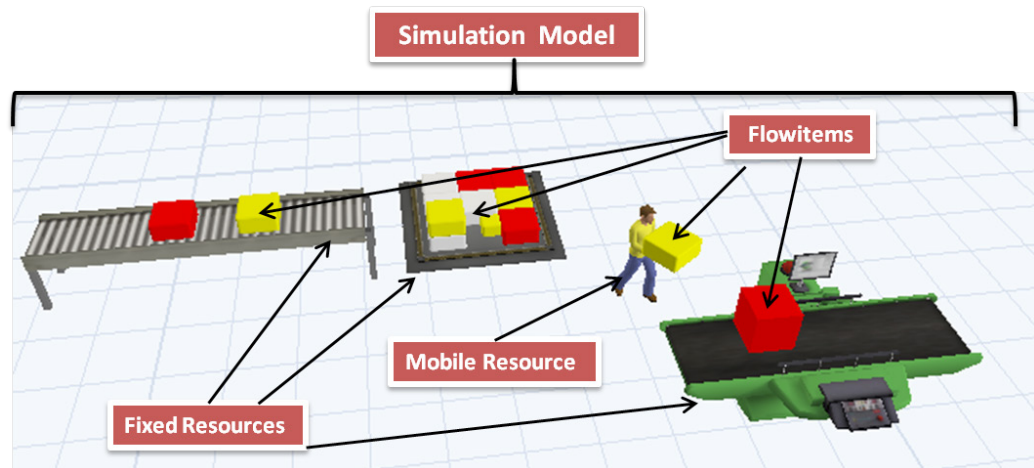


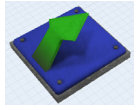
Figure 6.4 *FlexSim* simulation terminology

Section 6-3 Fixed resources

Fixed resources are the objects that send, receive, and perform activities/operations on flowitems. They are also the most common object type. They are referred to as “fixed” because they are largely stationary. Once they are placed on the model surface, they tend to stay in that place unless manipulated later by the modeler. Examples of this type of object include the following:

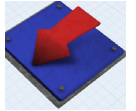
Objects that normally stay in one position and perform operations on a flowitem are called fixed resources.

Source



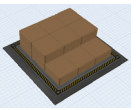
- Creates and releases flowitems
- Modes of arrival: interarrival time, arrival schedule, arrival sequence

Sink



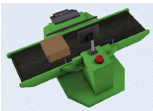
- Receives and removes flowitems from the simulation

Queue



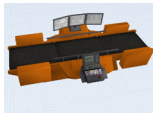
- Temporarily stores flowitems when downstream objects cannot accept them
- Can receive multiple flowitems at a time
- Can batch process flowitems
- Receives flowitems until its specified maximum content is reached

Processor



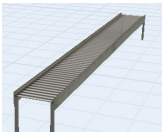
- Processes or forces a delay on a flowitem
- May call operators for setup or process operations
- May incur scheduled or unscheduled downtimes
- Handles one flowitem at a time or multiple flowitems independently (if its maximum content > 1)

MultiProcessor



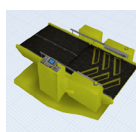
- Performs a set of operations or processes in sequence
- Operations/processes may have separate times and call separate resources
- Handles one flowitem at a time

Conveyor



- Moves flowitems over a fixed path (not necessarily linear) at a specified speed
- Flowitems enter and leave the conveyor one at a time
- Modes: accumulating, non-accumulating
- Capacity limited by number of flowitems or available space on the conveyor
- Spacing between flowitems may be specified

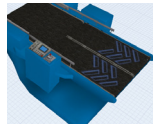
Combiner



- Groups multiple flowitems

- One input from Port 1 which may be a container to hold flowitems from other ports
- Can have process times, use resources, and incur scheduled or unscheduled downtimes
- Modes: join, cannot be separated; pack, container from port 1 holds others—can be separated; and batch, all objects travel as a group.

Separator



- Accepts one input at a time.
- Can have process times, use resources, and incur scheduled or unscheduled downtimes.
- Modes: unpack from packed container or split, makes copies of a flowitem.

Section 6-4 Transporting items

The transportation of flowitems in a *FlexSim* model is the physical movement from one resource to another. That movement may occur in a number of different ways. By default, fixed resource objects in a model pass flowitems to each other instantly. In addition, flowitems may move from one fixed resource to another through the use of an intermediary fixed resource object, such as a conveyor.



Flowitems may also be moved by *task executors*. These are special objects that have mobility within a model. As the name suggests, they are free to move about within the model executing tasks that have been assigned to them. They may transport flowitems or be used as a resource for fixed resource. Examples of task executors which move flowitems are a fork truck, crane, ambulance, wheelchair, or person. Task executors that move to fixed resources and perform other tasks include an operator required for some setup, processing, or maintenance operation.

All of the types of fixed resources, task executors, and other objects are shown in the library grid in Figure 6.5. The grid is located on the left side of the main *FlexSim* interface.

Flowitems, as well as all *FlexSim* objects, may carry user-defined data on them while the model runs. That data can be queried and/or set during different events. Future exercises will make use of this capability. This information is as versatile as the modeler desires. It can be used to help fixed resources make decisions in the model regarding process times and routing choices, in addition to gathering customized statistical information about the model. These object characteristics are referred to as *labels* in *FlexSim* and as attributes in other simulation software.

Task executors move flowitems from one place to another.

Data, used in simulation logic, can be placed on all *FlexSim* classes.

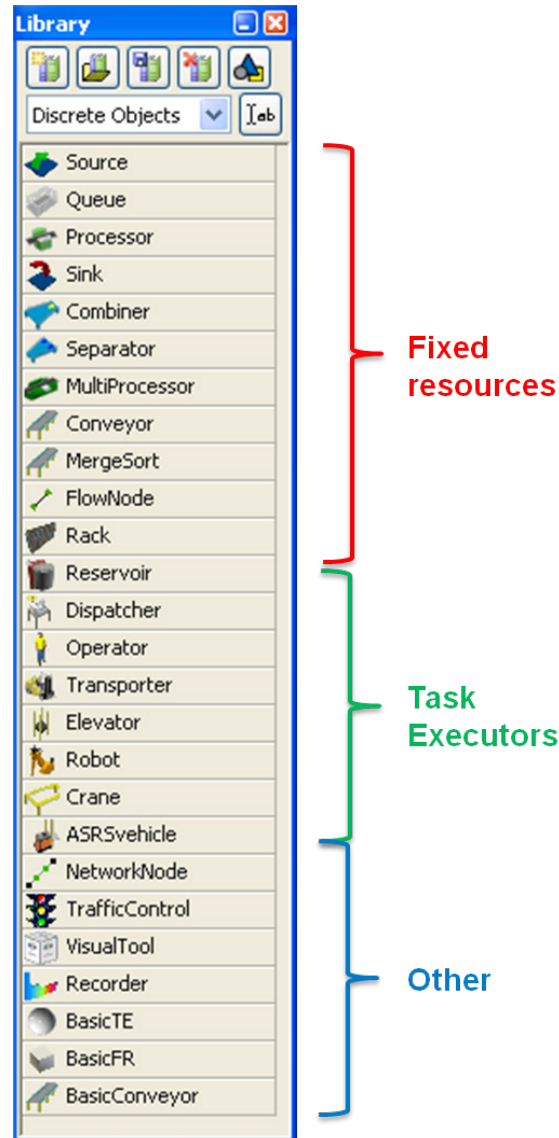


Figure 6.5 Library grid

Bringing objects to the screen

Objects that are used in a simulation model are usually moved from some repository accessed by a tool bar or drop-down menu. Various common conventions are utilized to move objects to the simulation window.



Objects are brought into the model from the *FlexSim* standard libraries or from custom-built libraries. The 3D objects in the library are generic and represent the functionality of the object; the images can be changed, as described in the Appendix.

Objects are brought onto the simulation surface (layout) in one of two ways:

1. Click and hold the left mouse button on the object in the Library, then drag it to the position in the model and release the mouse button.

Individual objects can be moved and changed in the 3D environment.

Editing object characteristics is achieved through the Properties window and its associated tabs.

2. Enter the create objects mode by clicking the Create Objects button  on the main toolbar. Then click the object to be created in the Library and click again in the model view where you want the object to be placed. Click in the Library again for another object.
3. When finished, click on the standard icon  (or Esc key) on the main toolbar.

Once objects are on the simulation surface they can be moved into position and changed visually in a number of ways. To move an object around on the surface of the model, click on it with the left mouse button, and drag it to the desired position. You can also move the object up and down in the Z direction using the mouse wheel while holding either the left or right mouse buttons down on the object.

To change the object's size and rotation, go to the main tool bar at the top of the screen and select Edit. Under the drop-down menu, click on Resize and Rotate Objects if it is not already selected.

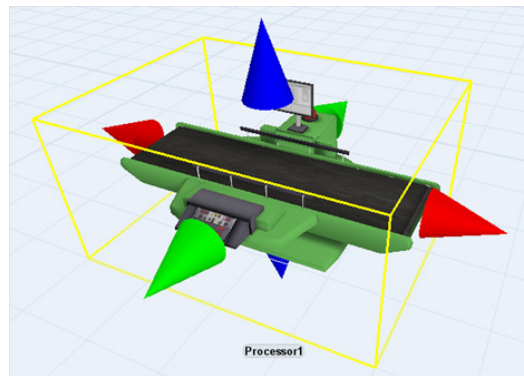


Figure 6.6 Changing an object with the mouse

As shown in Figure 6.6, when clicking on an object three colored arrows will appear along each axis. Note that a yellow box goes around the object to show that it is selected. Holding the shift key down and left clicking will result in a red box around the object. (see the appendix for details).

To resize the object, left-click on the desired axis and drag the mouse up or down. To edit the object's rotation, right-click on the arrow corresponding to the desired axis and drag the mouse forward or backward. To change the object's position on the Z axis, move the mouse scroll wheel. To reset the object, right-click on it and under Edit, select Reset Object.

Section 6-5 Editing objects

All objects in a simulation application have operational variables that can be modified to characterize their operation. Most graphical applications utilize a double click to open the object. The user interfaces and definition of the variables depend on the particular application. In this case, open *FlexSim*, select to build a new model, and drag out a processor object to follow along with the text.

Double click on the object to open it. This interface is used to define what the object does and how it behaves; for example, how long flowitems will stay inside the object, where they will go when they are released, and how the object will appear.

Properties window

The Properties window, as shown in Figure 6.7, is the default view when an object is selected by double clicking on it. At the top of this window is the object's name. As objects are created, they are given default names such as *Processor #*, where # is the number of objects created since the *FlexSim* application was opened. It is important that each object have a unique name.

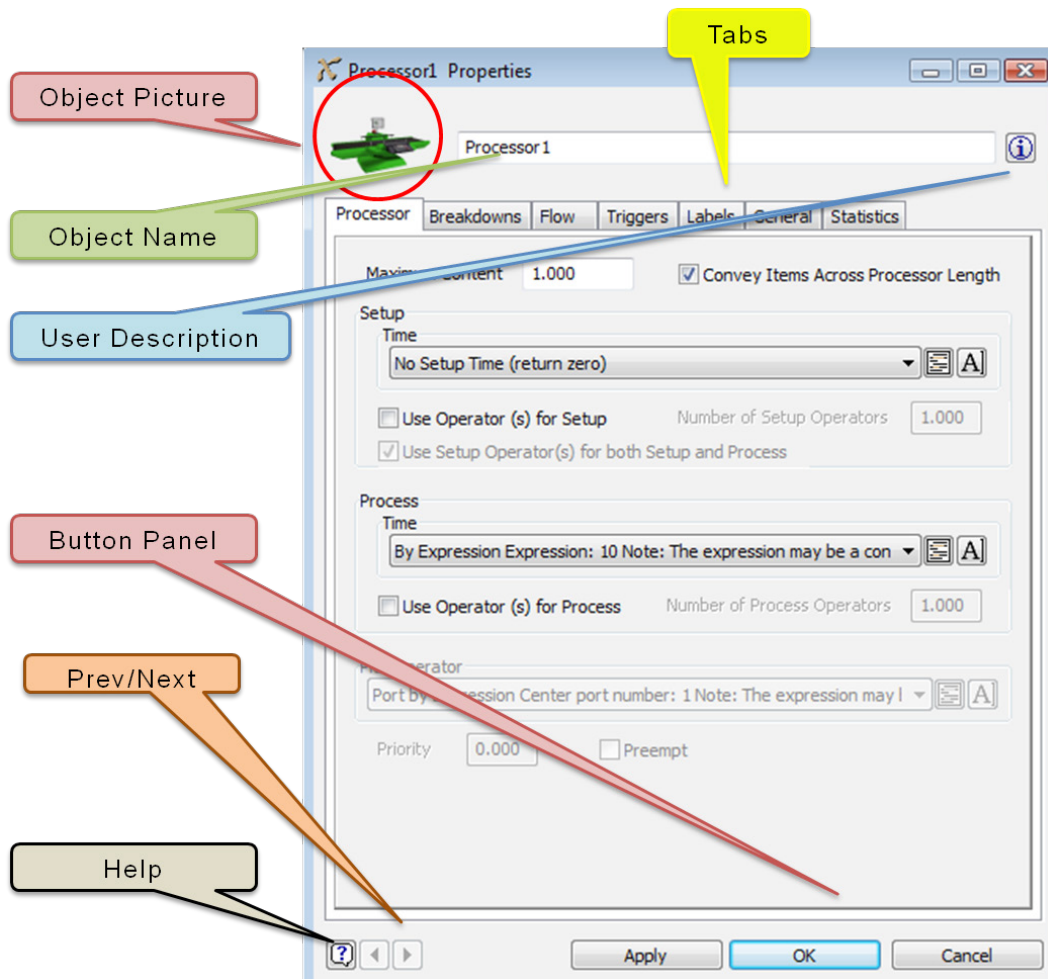


Figure 6.7 Processor's properties window

Having a naming convention can help as the number of objects in a simulation grows. One type of naming convention is indicating the type of object as part of the name; for example *pr_Machine1*, *qu_PartSurge*, *cv_OutputPath*—where *pr_*, *qu_*, and *cv_* indicate that the associated object is a processor, queue, or conveyor respectively.

Each object class (i.e., fixed resource, flowitem, task executor) has its own unique set of properties, although all objects will share a similar tabbed layout. The tabs open other windows that relate to a particular function. All objects have a General and Labels tab and most have Statistics, Flow, Triggers, and Breakdowns tabs. The common tabs contain the following information:

The properties tab is the default window for all objects and contains the most relevant information about the object.

- *General*: Attributes that affect the look, sizing, position, and rotation of an object; it also shows a listing of the port connections for the object
- *Labels*: Table of user-defined data or attributes associated with the object
- *Statistics*: Statistical information, specific to the object, is collected during a model run—including charts and graphs
- *Flow*: Logic for how flowitems move into or out of the object
- *Triggers*: Optional object functionality to enhance the object's behavior; create and respond to events involving an object
- *Breakdowns*: Information about the reliability of the object; data are used to set up time between failures and time to repair tables

After making changes in the Properties window, or any tab window, clicking *Apply* saves the changes to the object, but keeps the current window open. Clicking *OK* saves the changes and closes the window. Clicking *Cancel* or the *X* box in the upper right hand corner of the window closes it without saving the changes.

Within the object windows, some attributes are represented by a simple text editing field or check boxes while other attributes are represented by a drop-down list that *FlexSim* refers to as *picklist options*. The picklist itself is a drop-down list of common options associated with this parameter. Details for using the picklist options are contained in the Appendix.

General tab

The General tab window contains properties that impact the object's visual appearance as shown in Figure 6.8.

Items in the top half of the window modify the basic 3D shape of the object (details in the appendix). Just below the shape information is the selection window for the object's color. Objects that have been designed with user defined textures can change color. Clicking on the browse icon brings up a color selection window. Colors can be selected directly or by specifying the RGB values. When a new color is selected, click *OK*.

The lower half of the window sets the size, rotation, and position of the object with more precision than using the mouse movements described earlier. The values can be typed in directly or can be changed by using the up and down arrows. Clicking outside the cell after typing applies the change. Moving the windows around so that the object is seen at the same as the General tab window can make it easier to see the effect of any changes. Clicking on the Simulation window will bring that window to the front and may push the General tab window behind it.

Visual properties of an object are always found on the General tab.

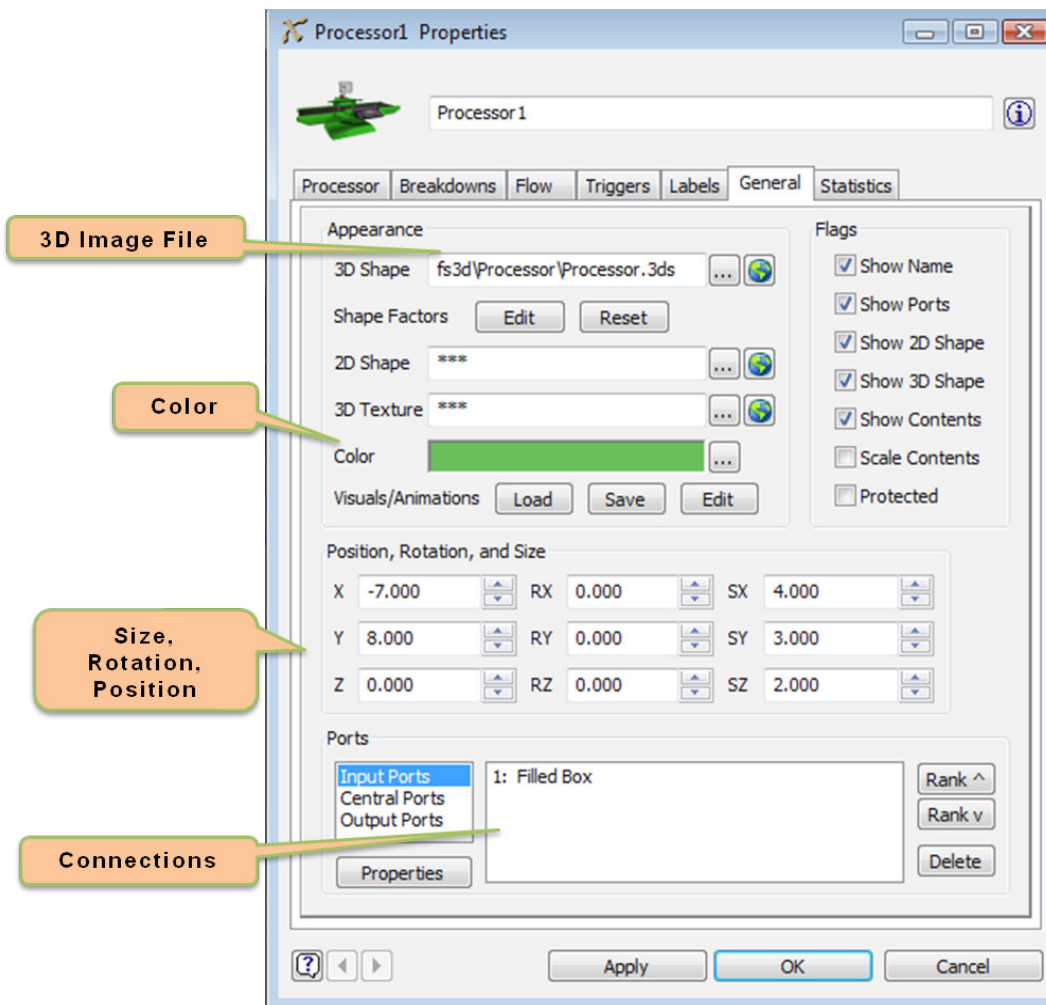


Figure 6.8 Object's General properties tab

At the bottom of the General tab window is information about the port connections of the object (port connections are discussed in the next section). Selecting the Input, Output, or Center Ports causes the objects connected to those ports to be listed. Port connections can be deleted by selecting them and then clicking on the Delete button. For some objects the sequence or order of the connections is important. That sequence can be changed by clicking on the connection and using the Rank up and down arrows to change the port order without having to redraw them on the simulation layout.

Statistics tab

The Statistics tab, as shown in Figure 6.9, provides information about the object that is collected during a simulation. These statistics include the throughput of the object as well as general statistics appropriate for the object. Some data are only available when the object is set up to capture specific historical data; this is discussed in Advanced techniques section.

An object is in one of 50 possible states at any time. The time in each state is maintained as a statistic.

Information about the object's behavior during a run is grouped on the Statistics tab in a standard format.

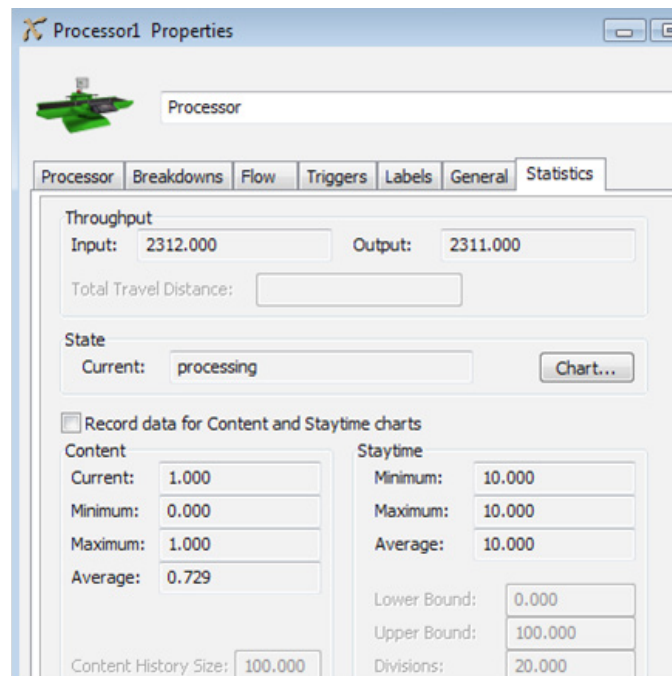


Figure 6.9 Object's Statistics properties tab

The Statistics tab also includes information about the state of the object. All operating objects are in some particular state at any point in time. That state is dependent on the type of object. As shown in Figure 6.10, *FlexSim* tracks 50 possible states during the course of a simulation.

Various statistics are developed based on those states. Only a subset of the total number of states may apply to any one object.

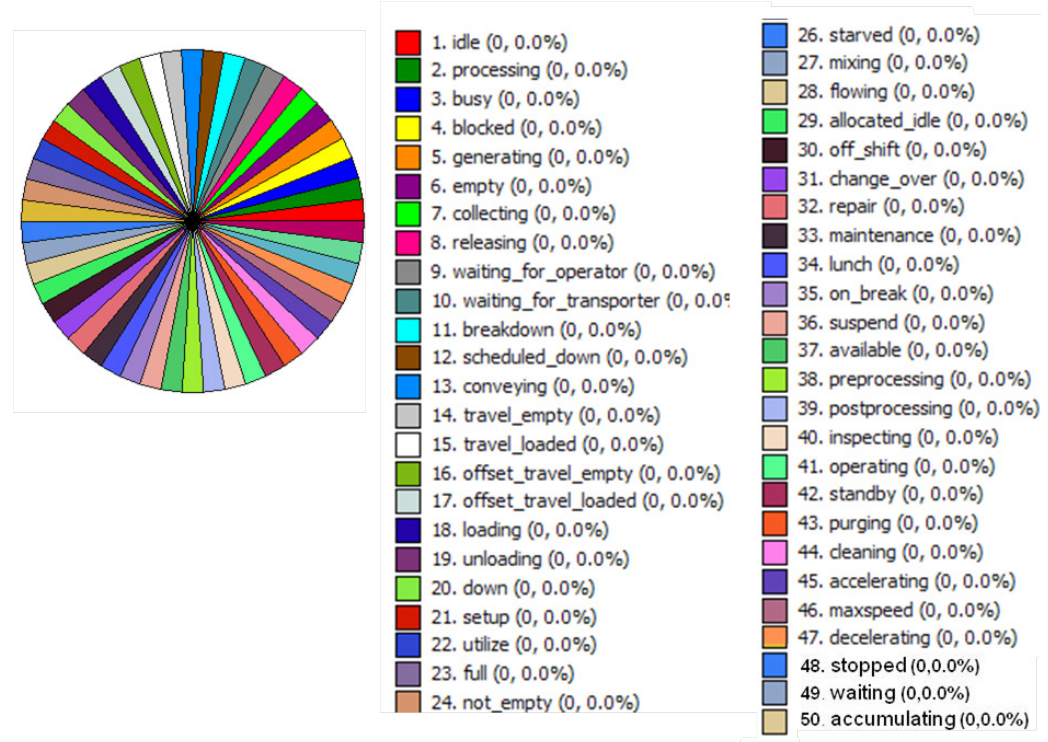


Figure 6.10 Object states

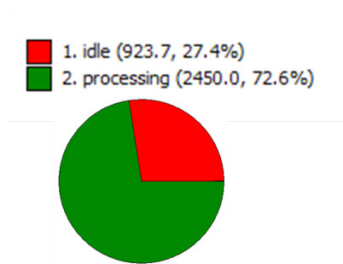


Figure 6.11 Object's state Pie Chart

Clicking on the Chart button in the Statistics window brings up a pie chart showing the amount and percentage of time the object was in any particular state. The window can be opened during a simulation to view how the states change over time. An example is shown in Figure 6.11.

Section 6-6 Making connections

All simulation applications contain a means of indicating how flowitems move through the simulation. The methodology may utilize drop-down menus or a drag-and-connect convention. The connections may be visible points on an object or software pointers. In all cases, correctly following connection protocols is critical to getting the model to function properly.

In *FlexSim*, fixed resources move or route flowitems through models using port connections. These port connections establish the relationships needed between objects to define a flow, as shown in Figure 6.12. Each output-to-input relationship defines a possible route between fixed resources. Every possible routing choice in the model has to have a connection that defines it.

Flowitems move from one object to the next through input and output ports.

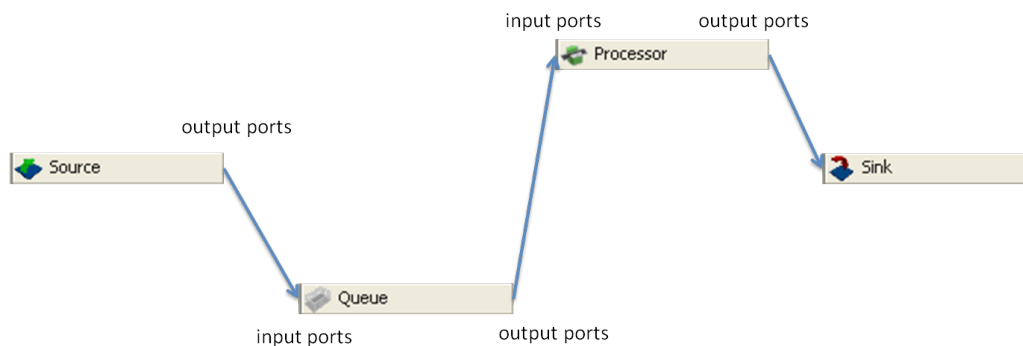


Figure 6.12 Routing of flowitems through port connections

Flow relationships in a model are defined in the direction of flow; that is from output to input. To create an output port to input port relationship in a model, connect the output port on the sending object to the input port of a receiving object. This connection is accomplished in the direction of flow by:

- holding down the A key,
- clicking on the origin or sending object,
- dragging the cursor to the downstream or receiving object.

When the mouse button is released a connection will appear. By default (unless otherwise specified)

- the movement from one object to another requires zero simulated time,
- the flowitem goes to the first available downstream object in the numerical order of the ports.

Figure 6.13 shows the input and output ports for a queue object. As a result of the connections made between ports, flowitems will move from the conveyor object to the queue object and then on to one of the two processor objects (to Machine2 via Port 1 or to Machine1 via Port 2).

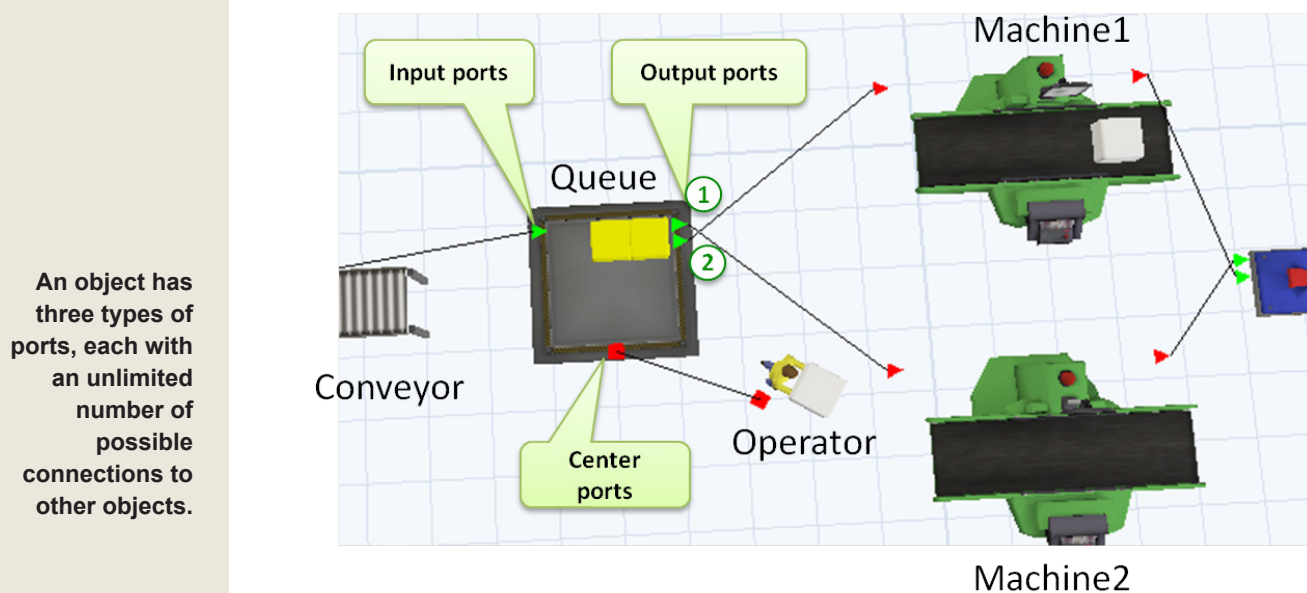


Figure 6.13 Object connections in a model

From the center of the queue is another connection to a task executor, in this case an operator. This connection is called a *center port connection*. A center-port relationship allows one object to reference and communicate with another object. A center port connection is most commonly used to specify a task executor for transporting flowitems between objects or to assist in an object's operation. Center ports are not directional and thus may be connected in any order. In summary, input and output ports direct the path of flowitems through a model; center ports facilitate communication among objects.

The red and green indicators at the ports signify their current condition as open or closed. For example, a processor will have a red input port indicator if it is busy and can't receive a flow item.

Input and output connections are deleted by holding down the Q key and dragging the cursor from the upstream to downstream object. Center connections are deleted by dragging between objects while holding down the W key. Connections

may also be deleted by going to the General tab on one of the objects involved and deleting it, as discussed earlier.

Objects can have an unlimited number of port connections.

Section 6-7 Moving flowitems

Once connections are created to indicate how flowitems are to move through the simulation, the exact way they move may be specified. In some applications simply making the connection defines the movement. In *FlexSim*, flowitems can move from one fixed resource object to another in a number of ways: immediately, via conveyor, via flow node, or by a task executor.

Immediate

When two fixed resources, such as a processor and a queue, are connected together, the flowitems “jump” immediately from one object to the other. The only visualization is the flowitem ready to leave one object and then suddenly appearing in the other.

Conveyors

The conveyor is used to move flowitems, but is itself a fixed resource since it stays in one place in a simulation. Double clicking on a conveyor opens an interface window specific to conveyors. There are many attributes that can be changed in order to specify how the conveyor should function. The main Conveyor tab is shown in Figure 6.14.

The left side of the interface directly impacts the conveyor’s operation. A few of the important attributes include the following:

- *Speed*: The speed of the conveyor is specified in grid units per time unit as discussed previously.
- *Maximum content*: The maximum number of flow items that can be on a conveyor at any point in time; however, this will only take effect if it is less than the length of the conveyor divided by the size of the flowitem plus item spacing.
- *Spacing Rule*: This rule sets how far apart items on the conveyor can be. The default is the size of the item.
- *Accumulating check box*: The check box in the Operation section of the window indicates if the conveyor is accumulating or non-accumulating. An accumulating conveyor, such as a one with rollers, allows material to enter at any time as long as there is enough space. Consequently, if the output end of the conveyor is blocked, items will still enter the conveyor on its input

Conveyors provide a standard way of moving flowitems from one object to another. They are used extensively in operations systems.

side and will accumulate until no more space is left or its specified maximum capacity is reached. A non-accumulating conveyor—such as one with slots or buckets—only allows one item on the conveyor for each slot. The entire conveyor will stop if material can't leave the conveyor, and no additional material will be allowed to enter.

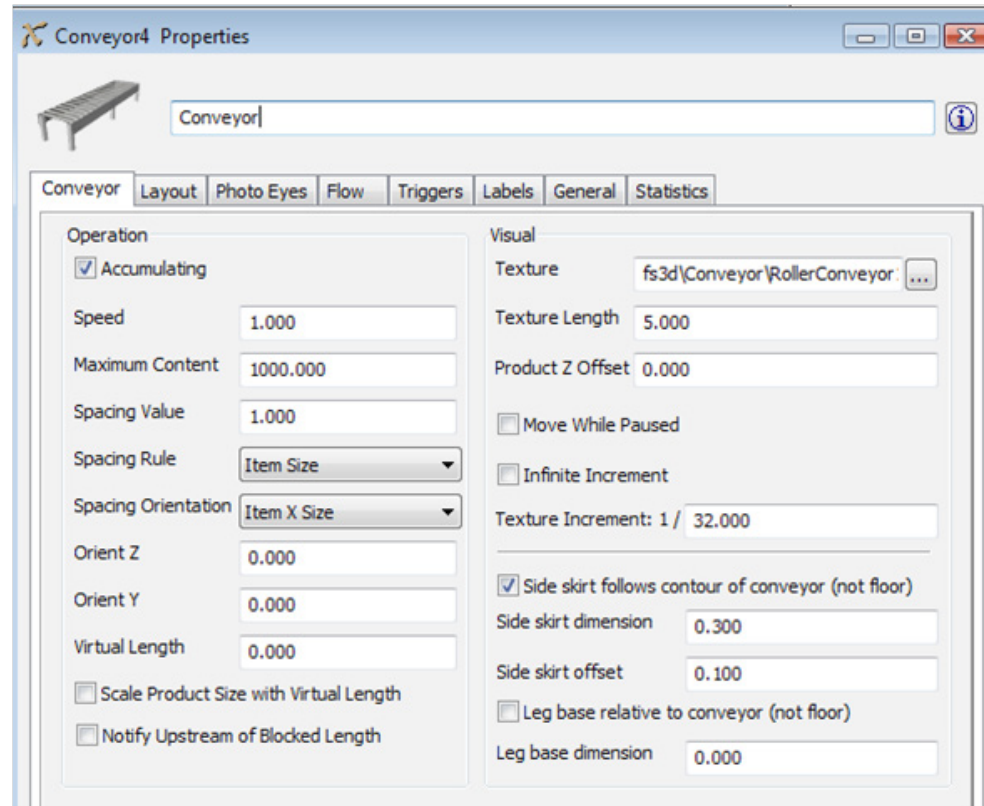


Figure 6.14 Main Conveyor properties tab

By default, the time it takes a flowitem to travel on a conveyor is the length of the conveyor on the screen divided by the speed of the conveyor. Another way to influence the conveyor time is to set a value in the Virtual Length field. If this field value is greater than 0, the time to convey will be the Virtual Length divided by the Speed.

The right side of the interface affects the visual representation of the conveyor on the screen. For example, the use of different textures or bit maps can change the conveyor's appearance.

Conveyors have two additional unique tabs: Photoeyes and Layout. Photoeyes are sensors that can be positioned along the length of the conveyor to trigger custom logic. Photoeyes are not covered in this section.

Conveyors can be configured to follow complex paths in the simulation by using the Layout tab. These complex paths can involve multiple changes in direction in all dimensions (X, Y, and Z). This is accomplished through the creation of

Conveyors can be structured to follow straight or curved paths.

multiple sections. Regardless of the number of sections used in a conveyor, it is still considered a single object. Details of modifying a conveyor layout are included in the Appendix.

Flow Node

Flow nodes are fixed resources that can act in a way that's similar to a very simple accumulating conveyor. Flowitems move along the output connections of a flow node but with no visible piece of physical equipment. The speed at which the flowitems move can be set, as well as the maximum content on the flow path. If the output of a flow node is blocked, flowitems accumulate at the end of the output connection path. Flow node connections are made in the same way as connections between other fixed resources—in the direction of flow holding the A key down.

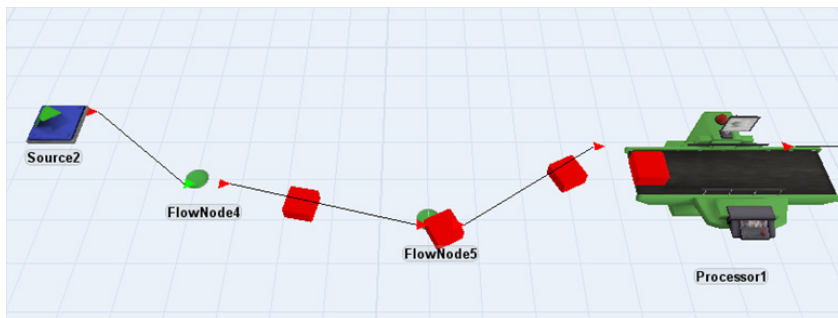


Figure 6.15 Flowitem travel using Flow Nodes

Task Executors

Task executors are mobile resources that can carry flowitems from place to place. They are covered in more detail in Chapter 7.

Section 6-8 Creating simple learning models

A good way to become familiar with the operating characteristics of any simulation software application and its objects is to create simple models or learning models. Such a model usually involves a source to create flowitems, conveyors to visualize the movement of flowitems, and any other objects as desired. Keeping the number of objects to a minimum is a good idea, as is using values that make observing results easier.

1. For example, to experiment with the operation of a processor, drag out a source, a conveyor, a processor, another conveyor, and a sink as shown in Figure 6.16. Open the source object and check the box marked Arrival at time 0. Also, select the By Expression option under the drop-down menu for the inter-arrival times. Use a simple constant value, such as 2, that will make looking at results easier. (If the time units of the model are in

Building a simple model that focuses on one object is the best way to learn about its operating characteristics.

minutes, then an interarrival time of 2 means 30 flowitems will arrive in an hour.)

2. Open the processor and choose By Expression in the process time drop-down, giving it a value of 1. Again, if the units are minutes, then the processing time is a constant 1 minute.
3. Open the conveyors and set Speed to 1.
4. Run the model and try changing variables.

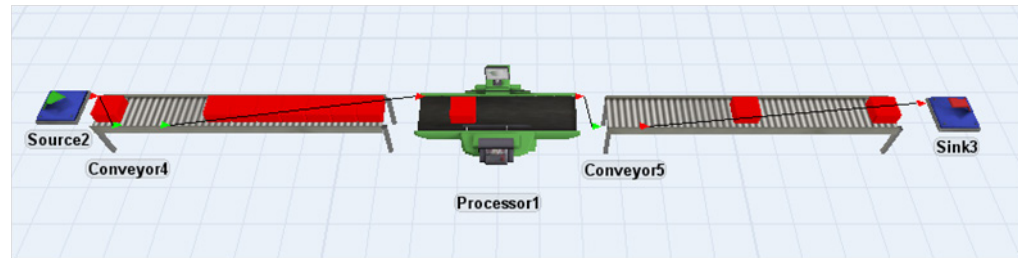


Figure 6.16 Simple model for learning and testing

Create other such simple models that include a minimum number of components and examine their behavior and responses to changes in logic.

Section 6-9 Single-run statistics

As discussed in Section 5, and shown in Figure 6.17, each object contains a Statistics tab that reports basic statistical measures as the simulation runs. Object performance measures may also be displayed on the model layout. It is common in simulation packages for the software to track, calculate, and report basic statistical information on objects in the model. In *FlexSim*, the primary object statistics that are displayed on the Statistics tab include the following:

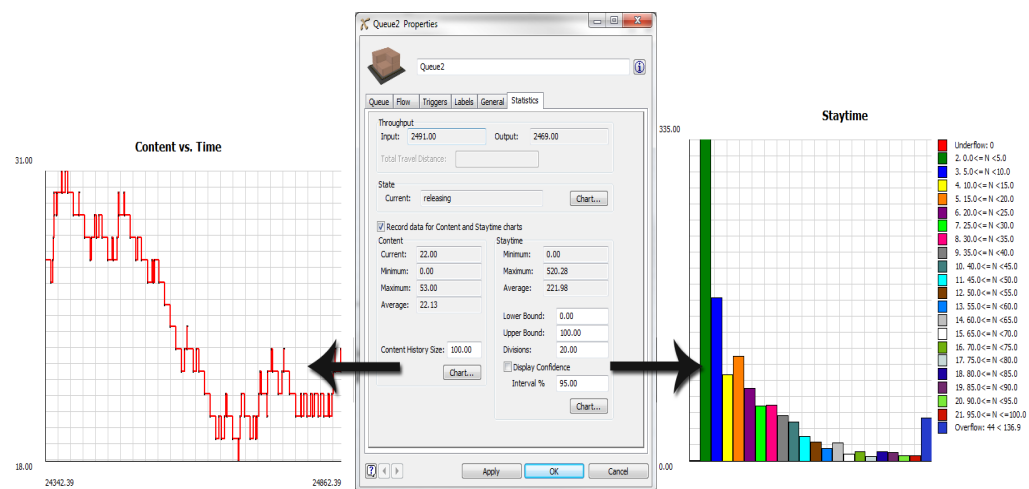


Figure 6.17 Object statistics

- **Throughput:** The number of items entering and leaving an object
- **State:** Current operating condition (e.g., idle, processing, waiting for transport), and a pie chart of the percent of time an object has been in each state since the start of a simulation run
- **Content:** Average, minimum, and maximum number of items in an object over the duration of a simulation run, and an optional line graph of the number of items in an object over time
- **Staytime:** Average, minimum, and maximum time that items spent in an object over the duration of a simulation run and an optional histogram of those times

These general measures are helpful for verifying that a model is performing as expected; however, simulation projects, as described in Chapter 4, should have specific objectives along with a corresponding set of definitive performance measures that are used to assess a system's performance. In other words, as a system is being designed and analyzed, there are a few key measures that are used to determine whether one configuration is better than another.

These key measures (also referred to as response variables), as shown in Figure 6.18, are the outputs from a simulation model. The outputs are compared to system performance criteria and drive the definition of alternative configurations and the corresponding values of the controllable input variables (also referred to as decision variables).

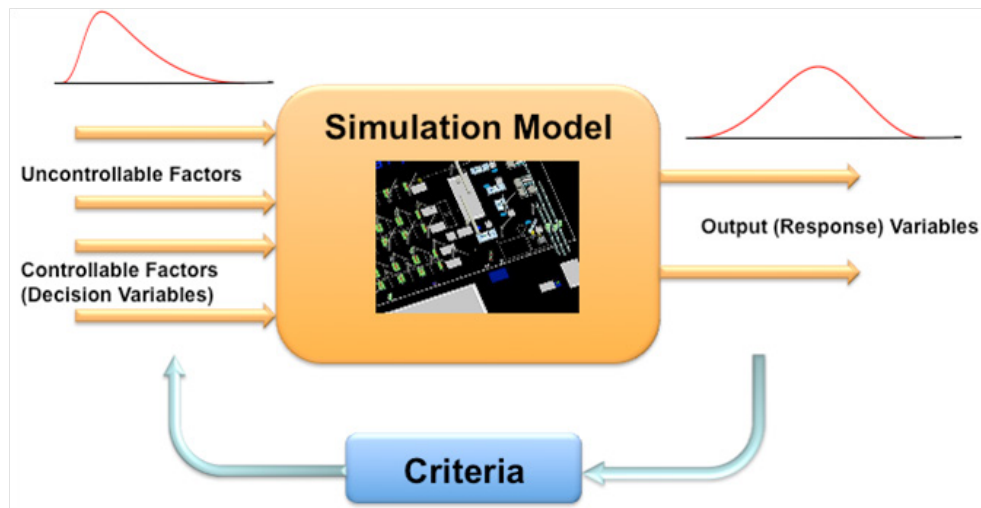


Figure 6.18 Stochastic model inputs result in stochastic model outputs

Figure 6.18 also indicates that if any input to a simulation model is stochastic or probabilistic, then the output will also be stochastic. Most of the time there are several random inputs to a model (e.g., time between arrivals, service times, time between failures, quality indicators). As a result, it is very dangerous to draw any conclusions from a single run of a model. Drawing conclusions from a single run would be analogous to, in basic statistics, making inferences based on a sample size

Objects maintain statistics over the duration of the simulation.



Never draw conclusions from a single run of a simulation model.

of one! Careful statistical analysis is therefore paramount when using output from a discrete-event simulation model.

In order to draw conclusions and make valid inferences, simulation models must be replicated; that is, the models must be run multiple times. This involves independent random samples from the various statistical distributions that can be used in a model. The information gathered from the multiple runs is combined to provide a basis for conclusions and inference. The process and methodologies for doing this are explained in Chapter 10.

Exercises

Simulation is an applied technology and has little meaning when used to simply create models without an objective in mind. The belief that if you just build a simulation something good will happen doesn't hold true in practice. Chapter 4 of this book emphasized the need to establish a proper scope and detail level for simulation projects and discussed a methodology.

Lessons learned by actually solving problems using simulation are more likely to be remembered. The exercises in this and later chapters follow this practice. In each case the problem background and associated data are provided. The steps to follow should include an analysis of the system, a definition of how best to simulate the system using an OFD for planning, and an understanding of what simulation functions are required to resolve the problem.

Exercise 6-1 Johnson Pharmaceutical

Background

As the largest supplier of over-the-counter medications, Johnson is redesigning its cold remedy line for higher capacity in time for the upcoming flu season. The plan is for the packing line to be completely re-designed. The new plan, conceived by the corporate packaging group, is based on the “greedy” concept. In that design, boxes of individual doses are taken off the main conveyor line by the first test/wrap machine.

A second test/wrap machine takes off boxes further down the conveyor. The boxes arrive in the packing area at an average rate of 60/min. The wrap machines are rated at an average of 40/min. Based on average rates, the first two machines can handle 133% of the incoming boxes rate; however, a third will also be added as a backup. The plant engineering manager has asked you to validate the design before the plant commits to the proposed production rate.

Problem statement

Validate the new packaging line design.

Operating data

Time between arrivals of medical boxes: exponentially distributed (0,1,1) with mean of 1 second. The first parameter is the location parameter and is usually 0; the second parameter is the mean; the third parameter identifies the random number stream to use (more on setting this value later).

- Test/Wrap machine cycle times: exponentially distributed (0,1.5,2) with a mean of 1.5 sec.
- Conveyor speed: 1 unit/sec.

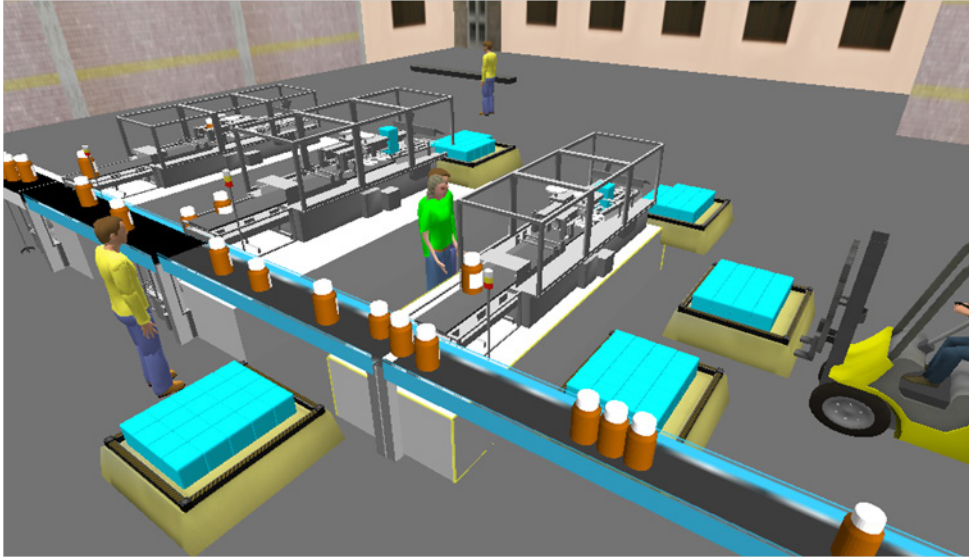


Figure 6.19 Pharmaceutical packaging

Expected results

- Draw an OFD for the system
- Comment on the design by running a simulation for an 8-hour shift (28,800 sec). What would you recommend to the plant manager? What rate should the plant commit to?

Modeling and analysis issues

- How should the conveyors be shown? What is the flow logic for them?
- Based just on the average rates what would you expect the capacity of the line to be and why?

Simulation results and comments are included in the Appendix.

Exercise 6-2 Lucky Air

Background

Lucky Air, a start-up airline, is committed to providing a shuttle service between the Orange County airport and the Las Vegas airport. With their fleet of regional jets, the company feels that the time is right to bring casino patrons for quick daily visits to Las Vegas. Their promise is to fly as long as there are people who want to travel. If a scheduled plane is full, another will be brought out; their motto is *Always a Winner*. They expect an increase in business as people try to find money from the slot machines during the downturn in the economy.



Figure 6.20 Lucky Airlines

The owner wants to set up operations as quickly as possible so he decides to operate their check-in counter with three ticket agents: one for passengers with e-tickets, another for passengers with paper tickets, and a third for passengers purchasing a ticket. As the only engineer in the new airline, you don't think the level of service will be good and the agents will be working inefficiently. Since you don't want to verbally confront the owner, you decide on showing what might happen with a simulation.

Problem statement

You believe the proposed operation will be inefficient and you want to illustrate the system's behavior to the owner.

Operating data

Estimates of passenger demand, in terms of times between arrivals to the ticketing area and time for an agent to service each passenger type are provided in the table below. Assume the time between arrivals is exponentially distributed while agent service time is normally distributed—all times are in minutes.

Passenger type	Time between arrivals	Service time
E-ticket	mean: 5 min	mean: 3 min std. dev. 1 min
Paper ticket	mean: 10 min	mean: 8 min std. dev.: 3 min
Purchase	mean: 15 min	mean: 12 min std. dev.: 3 min

Expected results

- Fill out Parts I and II of the project template.
- Provide the following metrics based on a simulation run of 168 hours:
 - The average and maximum length of the lines for each agent's station
 - The average wait time for each type of customer
 - The average number of customers each station services per hour (throughput)
 - The average utilization for each agent
- Prepare an executive summary based on conclusions drawn from the simulation.

Modeling and analysis issues

- How could the following objects be used to represent the waiting line?
 - a queue—what options would make it look like a line?
 - a conveyor
 - a flow node
- Could a processor be used as a conveyor? How? Under what assumptions?
- What are the differences in the four constructs identified above for getting passengers to the agents? Does the choice significantly impact the results?

Simulation results and comments are included in the Appendix.

Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the components included in the modeling environment of most simulation software applications.
3. Identify and define the basic elements that comprise a simulation model developed in *FlexSim*.
4. Describe how the movements of flowitems between fixed resources occur within a *FlexSim* model.

5. Describe the difference between fixed resources and task executors (the two basic categories of resources used in *FlexSim*).
6. Discuss how the *FlexSim* object structure enhances model development.
7. Select several *FlexSim* objects; for each one, identify the various states that the object can be in during a simulation.
8. Compare the behavior of accumulating and non-accumulating conveyors.
9. Construct a *FlexSim* model of the Compu-Help system described in the Chapter 5; however, assume the devices do not crash. Run the model for 10 hours and determine the following:
 - a. Average number of calls waiting
 - b. Average time a caller waits before being helped
 - c. Percentage of customers lost due to the system's limitation of only being able to keep six calls waiting.
10. Modify the basic Compu-Help model so that the maximum queue capacity is 1000. Run the model for 5 hours, stop it (do not reset), and note the average wait time. Run the model for 5 more hours, stop it (again, do not reset), and note the average wait time. Repeat this eight more times. Plot the average wait time versus cumulative run time. What can you infer from the graph?
11. Discuss how a model's behavior differs when a queue, conveyor, and flow node are used to move flowitems between two fixed resources.

Chapter

7

Adding Model Logic and Managing Data

The last chapter provided the basics for building discrete-event simulation models. It defined the functionality of some common objects, as well as some of the main properties of those objects. It also illustrated how to bring objects into the model and connect them so that the model represents a system of interest. This chapter explores some of the object properties in more detail and presents means to add logic to a model to enhance its representation of a system's behavior.

In the last chapter, information about an object's behavior was straightforward since each object was designed from the start to work in a particular way (e.g., a processor invoking a planned delay to represent service time). But what if the object's behavior changes based on the circumstances at the time, or if it changes when the value of certain parameters reaches a particular level? The person building the simulation model must decide where to best store object information so that it can be used appropriately to modify behavior or logic.

The choices and methods vary by simulation application, although the basic functionality is usually similar. In some cases the application itself may limit the choice; and in other cases, many options may be possible, and the decision is left to the model developer. At times the choice can be based on preference or design style, while at other times the choice is dictated by the system being simulated.

Section 7-1 Assigning attributes to objects

Most applications allow attribute information to be maintained on the objects themselves. This construct in *FlexSim* is called a Label. Attributes or labels, provide a means to associate specific properties with an object or to maintain information about an object. Labels are created on objects by opening the Labels tab on the properties window of an object.

There are various ways to add data to objects or flow items.

Labels can be either numeric or string (they can be arrays as well, but that is an advanced topic). Labels can be referenced from anywhere in a simulation model or at any time during a model's execution. The values can be set manually or changed dynamically. Oftentimes, labels are used to store numeric “flags,” indicator variables that change dynamically and indicate the current state or position of an object or flowitem.

Additionally, flowitems have a special default label called its itemtype. It is a numerical variable that can be changed dynamically. Itemtypes can be used to denote different types of flowitems within the same flowitem class. For example, if the flowitem class represents products, then their itemtype can be used to differentiate small, medium, and large products. Similarly, if the flowitem class represents patients, then their itemtype can be used to denote patients with appointments, walk-ins, emergency cases, etc. An itemtype can be used to set the value of other attributes, such as color and size, enable routings, and affect other types of logic and behavior. Creating labels on flowitems is accomplished through the Flowitem Bin as detailed in the Appendix.

Section 7-2 Adding logic

A simulation model is primarily a collection of various types of logic that, when combined, appropriately and reasonably represents the behavior of the system of interest. A large amount of default logic is automatically included in a model when basic objects are added to the simulation view surface and connected to each other. The inclusion of default logic is one of the main features of specialized simulation software. This predefined logic represents common system behaviors and is quickly and easily available to the modeler. Of course, since every system is different, simulation software also provides the means to easily modify the logic to more fully represent the system being considered.

Most of the remainder of this book presents ways to change and enhance model logic. This section begins that discussion by introducing the use of triggers found on an object's Trigger tab. In general, triggers are procedures that cause a specified behavior. They are invoked in response to a certain event or when a condition is met. For example, they are used to take a specified action (e.g. change an object's color or set a label value) when a flowitem enters an object, set a process time when a flowitem is ready to be processed, direct a flowitem to another object based on certain conditions (e.g. the value of the flowitem's itemtype or which downstream object is available), etc. Triggers may cause events to occur. For example, a downtime trigger may cause an object to become unavailable or a message trigger causes an object to respond to information contained in the message.

Triggers are prevalent throughout *FlexSim* and *FlexSim* provides many built-in trigger options, such as set color or set a label value. *FlexSim* also provides the capability to modify prebuilt triggers or create completely custom triggers.

Triggers tab

Triggers are action points that execute a set of logic when a particular event occurs, such as a flowitem arriving at, or exiting from, an object. All objects have triggers, but some have more than others depending on the object's functionality. When a trigger occurs, the logic associated with it fires, and is executed.

Most of the triggers associated with an object are contained on that object's Triggers tab. Figure 7.1 shows an example of a Triggers tab. The names, such as OnReset, describe the event when the trigger procedure is executed

Triggers are provided at standard events when logic may be needed.

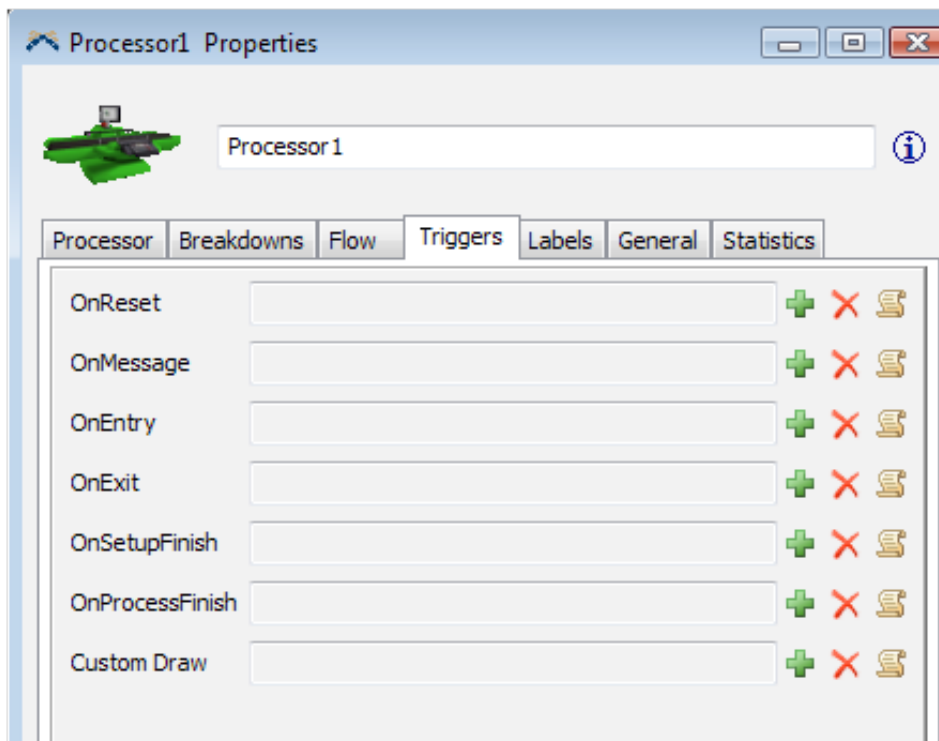


Figure 7.1 Types of triggers available on a processor

Common trigger points that are available on most objects include the following:

- *OnReset*: Occurs whenever the simulation model is reset; this is a good place to establish initial conditions for variables and labels
- *OnMessage*: Occurs whenever a message is sent to the object; messages may come from other objects to coordinate logic or from the same object to create a delay time or initiate logic at some future time
- *OnEntry*: Occurs whenever a flowitem enters the object; this is often used to change flowitem label values, color, or itemtype
- *OnExit*: Occurs whenever a flowitem leaves the object; this is oftentimes used to update data or take other action before the next flowitem enters the object.

- *Custom Draw Code*: Occurs each simulation time cycle and allows custom 3D shapes to be added (e.g., changing an object's appearance depending on its current state)

Triggers that relate to processing, collecting, loading, or other functions appear on the appropriate objects. The desired actions that occur at a trigger event are set by the drop-down menu options for each trigger type and are detailed in the Appendix.

Processing (or planned delay) time triggers

A flowitem triggers an event when it is subject to a planned delay, such as processing or setup operations. Processing times are therefore set through triggers, found on the Properties window of an object.

Typical events for a processor object include setting the setup time and/or processing time. Other events are triggered within a process: first, when an item enters a processor and there is setup time involved; second, when an object is ready to process a flowitem; and third, when the object has completed its processing of a flowitem and is ready to perform another activity.

Processor tab picklist options

As shown in Figure 7.2, processing times can be set by expression (e.g., a constant 5 minutes or a calculated value based on object properties), by sampling from a probability distribution, by basing it on label value, etc.

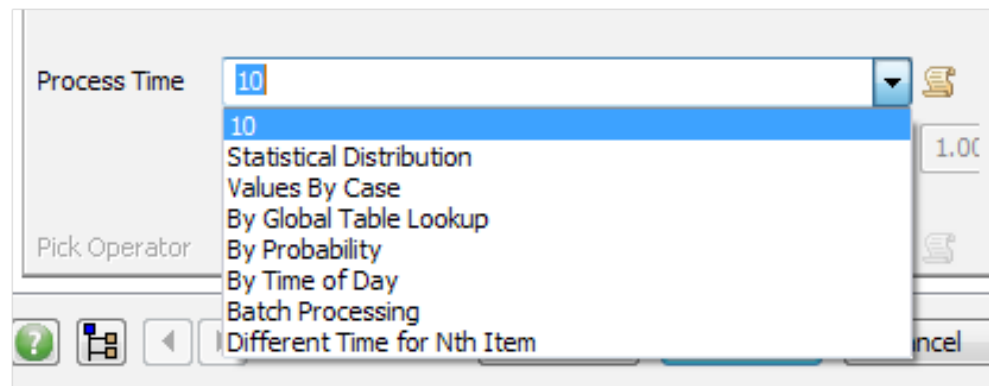


Figure 7.2 Processing time standard choices

Flow triggers

Other triggers are located on the object's Flow tab. As the name indicates, the Flow tab focuses on the movement of the flowitem from the current object to the next object. As shown in Figure 7.3, there are two sections on the Flow tab: Output (which is always active), and Input (which is only active when Pull is selected).

Flow triggers describe the logic for directing where flow items will go to next.

- Output
 - *Send to Port:* Specifies the logic that determines through which port a flowitem can travel after it leaves the object
 - *Use Transport:* When selected, calls a task executor to transport the flowitem to the next object; the drop-down menu is used to specify which task executor should be used
- Input: The Pull option is used when items should be pulled into the current object rather than being pushed from the previous object (default method).
 - *Pull from Port:* Specifies which port or ports should be used for the pull logic
 - *Pull requirement:* Specifies the logic for pulling a flowitem by some criteria, such as by itemtype or a label value

The example Flow tab in Figure 7.3 shows a processor object that pulls flowitems from preceding objects and, after processing is complete, sends flowitems to the first available downstream object; a transport or mobile resource is used to move the flowitem to the selected object.

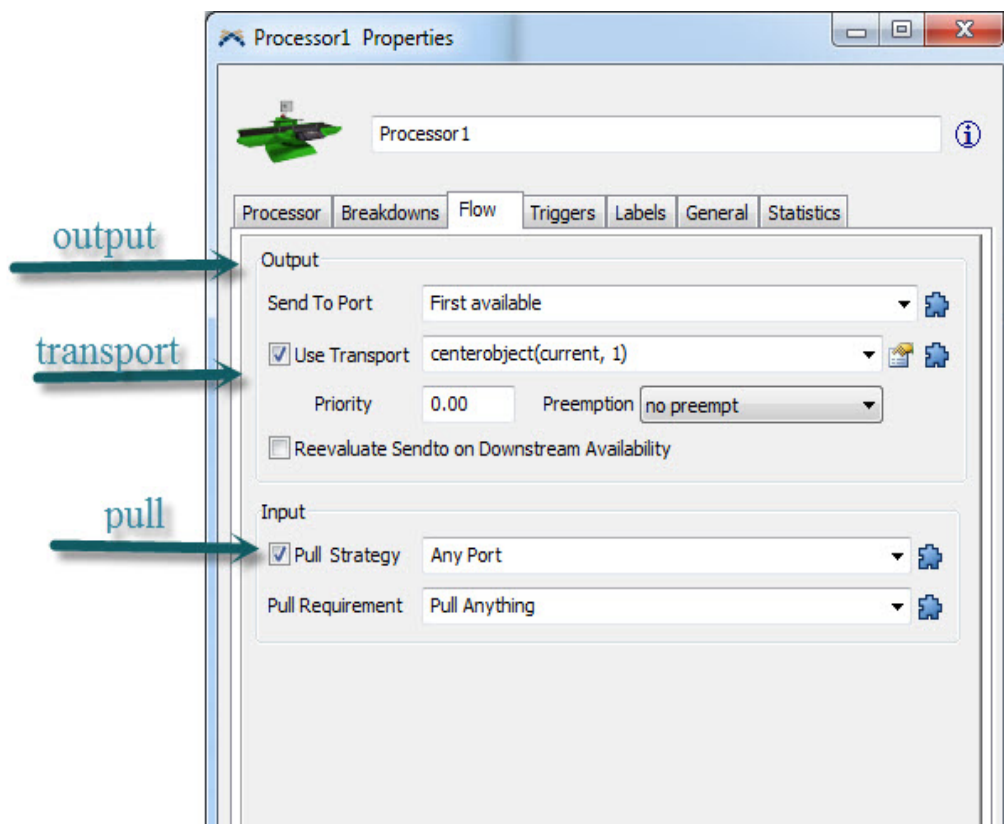


Figure 7.3 Example Flow tab with transport and pull options selected

By default, flow items will move to the next available object and will not be pulled.

The default settings for the Flow tab for objects are as follows:

- *Send To Port* picklist: First available Open all ports
- *Use Transport check* box: not selected
- *Pull check* box: not selected
- *Input*: pushed from the previous object

In Output, selecting First Available Port as the Send To Port trigger means that a flowitem will normally exit via Port 1, but if Port 1 is not available, the flowitem will exit via any other ports that are connected; however, the port evaluations are made in sequence. By default, no transport is used to move the flowitem to the next object, and the next input to be processed is pushed from the previous object. Input is selected based on the order of the port connections; that is, the object tries to take in a flowitem from Port 1 first, then if none are available, it tries to take in a flowitem through Port 2, etc.

The way objects are connected is related to the logic used in the flow triggers. As shown in Figure 7.4, there are various pre-built options for both Send to Port and Pull Flow logic. For example, some of the ways that flowitems can be sent to downstream objects are based on the shortest queue, in a round robin fashion, randomly, according to a specified probability distribution, based on a flowitem's label or itemtype, etc.

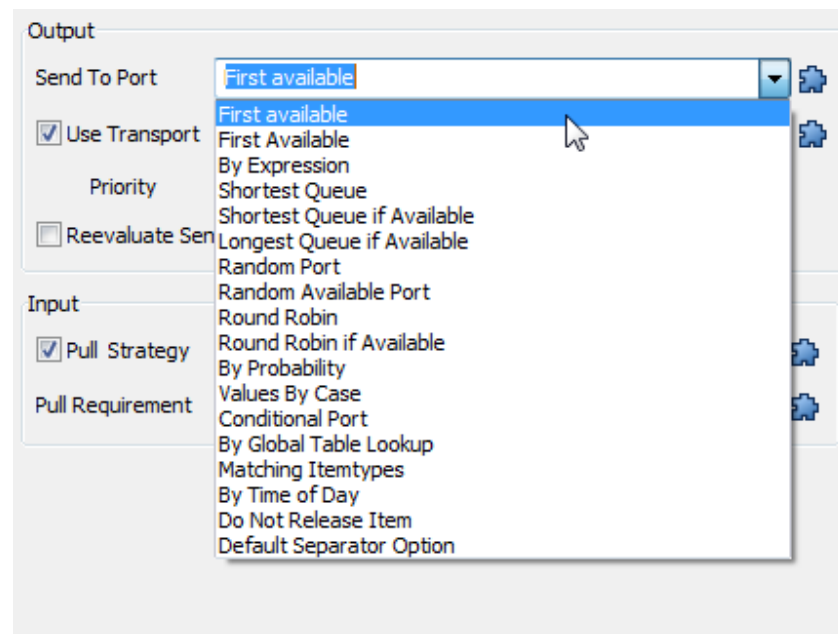


Figure 7.4 Standard choices for flow logic

Similarly, flowitems can be pulled from preceding objects based on the longest incoming queue, in a round robin fashion, randomly, according to a specified probability distribution, etc. In addition, custom logic can be developed using the code icon as discussed in a later chapter.

Pre-defined logic covers the most common means for representing item flow.

Figure 7.5 summarizes the trigger actions discussed so far, which are available in many objects. In this case, once an object completes the processing of a flowitem, it decides which port to send the flowitem through by executing the Send To Port trigger. It then executes the logic required to obtain a transporter to move the flowitem to the downstream object—if the Use Transport box is checked. As the flowitem leaves the object, any logic in the OnExit trigger is executed. The object is then available to process another flowitem from an upstream object.

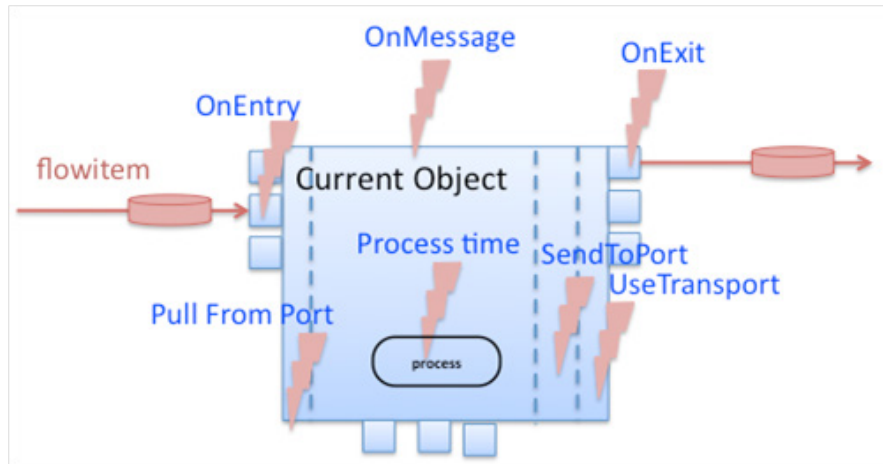


Figure 7.5 Types of object triggers

If the object pulls from an upstream object, the Pull From Port logic determines which port to use and thus which upstream object to pull from. As the flowitem enters the current object, logic in the OnEntry trigger is executed. The object then executes the logic in the Process Time trigger to determine how long to spend processing the flowitem. Finally, an object may get a message from another object at any time; when that occurs, the logic in the OnMessage trigger executes.

Other triggers

There are other types of triggers that can be used to invoke logic. Similar to the processing time trigger discussed above, the time between arrivals in a source is also a trigger. Arrival times are based on simple logic, like a constant (by expression), a sample from a specified probability distribution, etc. An example of modifying the trigger logic on a source object is detailed in the Appendix.

Also, events can be created by triggers that change the state of an object when there is a planned downtime on the object (e.g., preventative maintenance, breaks and lunches for operators, shift schedules) or an unplanned downtime on the object (e.g., machine failure or defective parts). These types of triggers and the resulting events are discussed in a later section on time tables.

Triggers are also used to create events that are more global in nature.. For example, triggers can invoke logic when a model is opened or executed, at different times

Object triggers fire at different points in the operation.

Triggers are consistent over various objects.

during experimentation (this type of trigger will be discussed in a later chapter on the topic of experimentation), and through general-purpose triggers called User Events.

Exercise 7-1 More Lucky Air

Background

The president of Lucky Air was not pleased with the information and analysis that was obtained from the last simulation and wanted to try to level the workload for the check-in agents as well as help the customers. He liked what he saw at his local bank, where there was one line of customers that then split to any of the available tellers.



If implemented at Lucky Air, this will mean that the check-in agents all will have to be trained to handle each type of ticket; therefore he wants to know the impact on service if this change is implemented.

Problem statement

Considering a single customer line with three agents who can handle all ticket types, will performance improve for the customers and balance the agents' workload?

Operating data

The operating data are the same as presented in the Lucky Air exercise in the previous chapter but is repeated below for convenience.

Passenger type	Time between arrivals (exponentially distributed)	Service times (normally distributed)
E-ticket	mean: 5 min.	mean: 3 min., std. dev. 1 min.
Paper ticket	mean: 10 min	mean: 8 min., std. dev. 3 min.
Purchase	mean: 15 min.	mean: 12 min., std. dev. 3 min.

Expected results

- Revise the OFD to match the proposed changes.
- Use a conveyor to represent the single waiting line; review section 3 of the Appendix for Chapter 6 to make it look like a moving walkway.
- Use the simulation model to provide the following metrics based on a simulation run of 168 hours.
 - Average and maximum length of the waiting line.
 - Average passenger wait time.
 - Average number of customers each station services.
 - Average utilization of each agent.

Modeling and analysis issues

- How will you identify the passenger types?
- Where should the service time for each type be located? Consider the following options:
 - Service time stored with each passenger
 - Service time stored on the agent
- What triggers on the source should be used?
- Where should passenger data be kept? How would the logic change depending on where the data is located?
- How will the connecting of the walkway to the agents impact the agents' workload?
- Once the passenger is done being serviced by an agent route the passenger to one of three sinks to keep track of the number of each type of passenger.

Details for building the simulation are contained in the Appendix.

Section 7-3 Managing data tables

Since simulation models are data-driven, managing the model's data is an important issue. One important practice is to remove the data from the internal logic of the model so that it can be easily changed for sensitivity analyses and manipulation by less experienced users. In *FlexSim*, one convenient way to store data, both input and output, is via a Global Table object. These table objects can be read from, and written to, during the execution of the simulation model.

Since global tables are needed in the next exercise, one will be setup here as an example. The context on how the table is used in the example is explained in the Appendix.

Global tables are added and edited through the Global Tables option on the Tools menu, which is part of the main Menu bar. Choosing the Add option creates a simple single-row, single-column table that can be modified by specifying the number of rows and columns in the appropriate boxes. The table for the next exercise is shown in Figure 7.6 with the basic table structure shown first, followed by the completed table.

Note that editing the header cells (Row 1, Row 2, Column 1, Column 2, etc) has no impact on the data; these cells are headers for information purposes only. Although optional, they enhance the readability of the model. Copy and paste functions and movement between cells is similar to Microsoft *Excel*. The table can be moved to *Excel* by highlighting the table and using the copy and paste commands. Data can be moved

Global tables provide a location for data that is easily accessible throughout the simulation.

into the table from *Excel* in the same way. Data can also be automatically exchanged between *FlexSim* tables and *Excel* worksheets; this will be discussed in a later chapter.

In the example in Figure 7.6, the table is named using the object-naming convention discussed in the Appendix for chapter 6: the prefix *gt* for Global Table followed by a descriptive name, in this case, PassengerTypes. As with other objects, the table name should be unique for the model.

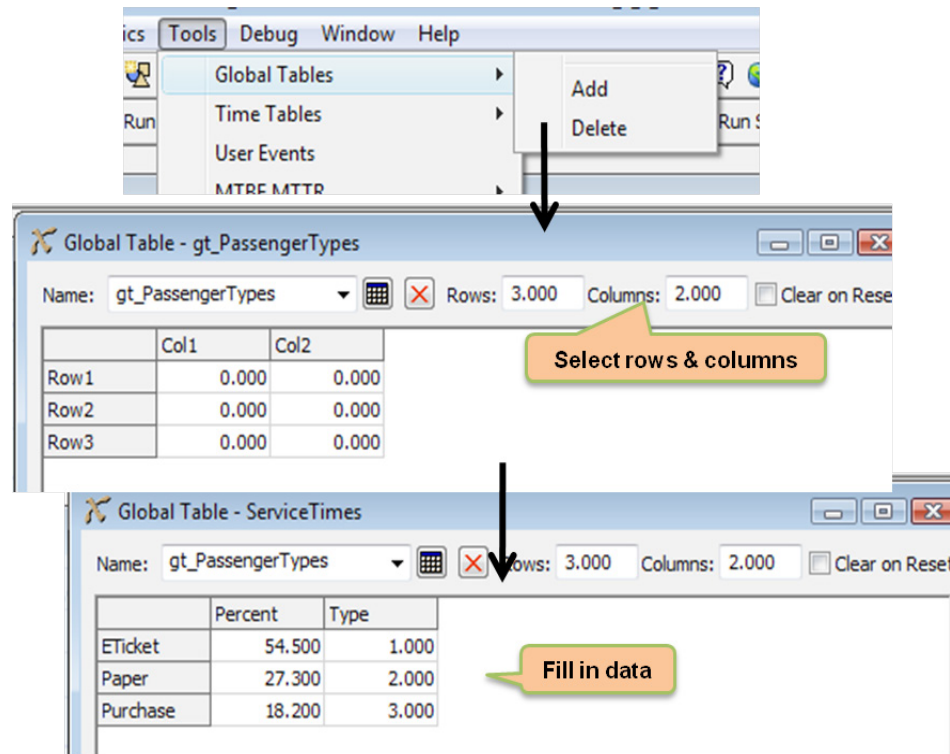


Figure 7.6 Building a Global Table

Exercise 7-2 Even more Lucky Air

Background

Lucky Air has lived up to its name, and many of its patrons have become high rollers at the casinos. The casino operators will provide Lucky Air with a subsidy for each high roller that they bring in. Consequently, there's a big push to keep the high rollers happy so they keep flying on Lucky Air. The head of the airline wants special services for high rollers, especially when they check in.



Problem statement

Prepare a simulation model that will show how the high rollers will be treated if there is a separate priority line for them to be served by the agents.

Operating data

The data are the same as in the previous Lucky Air exercises with the following exceptions:

- It is estimated that 10% of all passengers will be high rollers regardless of the type of ticket they have. Agents will give priority to any passenger in the high-roller waiting line.
- Assume that you didn't know the individual passenger arrival rates but only knew the average total rate (22 per hour) and the percent of each type of customer (54.5% E-Ticket, 27.3% Paper, 18.2% Purchase).

The following arrival schedule table can be set up based on the data provided above. For example, if on the average, E-ticket passengers arrive every 5 minutes, then their mean arrival rate is 12 per hour. Performing this calculation for the remaining customer types results in a total arrival rate to the system of 22 per hour. Therefore, the average time between arrivals is $1/22 = 0.045$ hr. or 2.7 minutes. On the average, E-ticket passengers comprise 12 of the 22 total passengers per hour, or 54.5%

Arrival information

Passenger type	Time between arrivals, minutes	Arrival rate, passengers per hour	Percentage of total
E-ticket	5	12	54.5
Paper	10	6	27.3
Purchase	15	4	18.2
Total		22	100.0

Assume the time between arrivals is exponentially distributed. Technically, this is not correct since the variable that is the sum of three different exponentially distributed random variables is not exponentially distributed; however, for now, this is a reasonable assumption.

Use a single source for the customers and determine the passenger type using the dempircal function.

Expected results

Use the simulation to provide the following metrics based on a simulation run of 168 hours:

- Average and maximum length of the waiting lines for regular and high-roller passengers.
- Average passenger wait time.
- Average number of customers each station services.
- Average utilization of each agent.

Modeling and analysis issues

- What trigger and which drop-down command should be used to set the itemtype?
- With the itemtype placed on the item, where is the service time logic?

Chapter 7 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Describe the different ways that attributes can be assigned and stored on a flowitem.
3. Define trigger. Select four triggers, each on a different object. For each trigger, explain when it will occur, describe one action that it can perform, and provide an example of how it would be used in a simulation.
4. Explain the function of each of the four icons associated with triggers on the Trigger tab.
5. Explain the action that occurs when the default Send to Port flow trigger on an object is executed.
6. Indicate the actions taken by the flow trigger options “round robin” and “round robin, if available.”
7. Describe a situation in which each of the following options could be used. They are options available on the Send to Port flow trigger.
 - a. Shortest queue
 - b. Random
 - c. Round robin
 - d. By Percentage
 - e. Conditional
8. Describe a situation where the “pull” option on the Flow tab may be used.
9. At what point in a Processor object’s activities is the decision made to send a flow item out through a particular port?
10. Define the steps that are required to build a global table. Describe how data might be exchanged with Microsoft *Excel*.
11. Describe the advantages and disadvantages of storing information on an object or flowitem or in a data table.

Chapter

8

Managing Entities and Time Tables

This chapter extends the Intermediate User's modeling capability in several key areas. In many situations items need to be grouped and ungrouped as they flow through a model - this is discussed in the first section. Mobile resources are one of the main classes of objects in any simulation software. In *FlexSim* mobile resources are referred to as task executors; these are especially versatile and powerful objects. The second section of the chapter describes the various types of task executors including their basic operation and the main properties that drive their behavior. In order to enhance visualization additional information oftentimes needs to be displayed on the simulation surface; this information may include statistics, state status, textual descriptions, etc. Section 3 describes how this information can be displayed in *FlexSim*. The final section describes the use of time tables to control object states. The chapter uses several comprehensive examples to illustrate these concepts.

Section 8-1 Grouping and ungrouping flowitems

There are many situations in operations systems in which the items flowing through the system are grouped or ungrouped. Examples include packing items in a tote for handling or in a box for shipping, unpacking parts from suppliers for assembly, unpacking totes after movement, making multiple copies of an item and distributing them in the system, etc. In *FlexSim*, these operations are handled by the fixed resources known as *combiners* and *separators*. Combiners and separators share many of the standard attributes of other fixed resources, especially processors, but have some specialized features of their own.

Grouping and ungrouping items are common events in a simulation.

Combiner

The combiner object, as the name suggests, combines flowitems. A unique tab on the object is the Combiner tab, as shown in Figure 8.1.

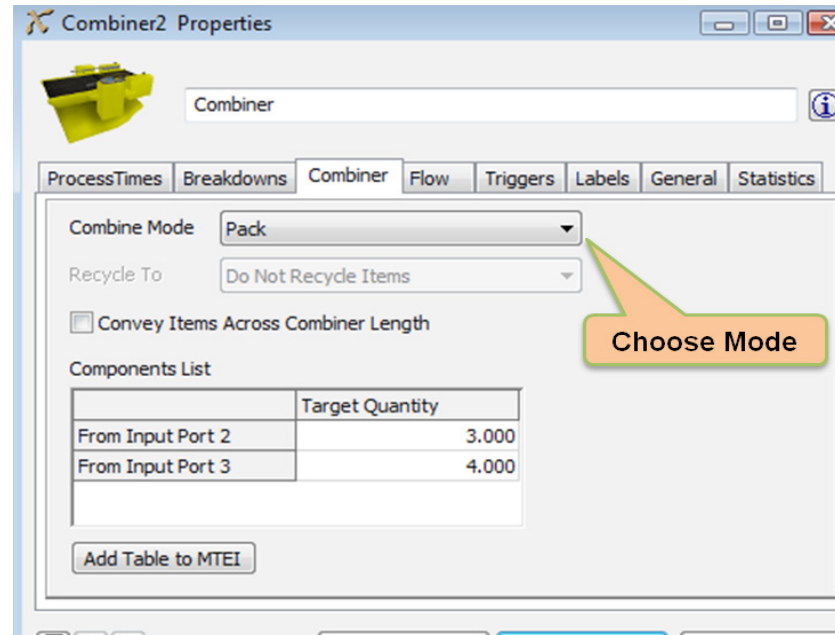


Figure 8.1 Combiner tab using the pack option

There are different ways that items can be combined.

The input connected to the first port of the combiner takes on special meaning depending on the type of combine function that is selected. The item entering from the first port is referred to as the *container object* and oftentimes represents a pallet, tote, or box. Other flowitems are placed on or in the container and enter through the other input ports.

The following are general characteristics of the combiner:

- A flowitem from Port 1 is required for the combiner to function.
- The target quantity to be combined from Port 1 is always 1.
- Quantities from other ports are specified in the Components List table on the Combiner tab.
- Flowitems from Ports 2 and higher will not be collected until a flowitem enters through Port 1.
- Processing time starts when all flowitems are collected.

Items may be either permanently or temporarily combined. The example in Figure 8.1 will pack (temporarily group) three flowitems from Port 2 and four items from Port 3 in the container that enters the combiner through Port 1.

The picklist options for combining flowitems found on the Combiner tab of the object are summarized below:

Pack

- The flowitem from the first port is designated as the container.
- Flowitems from other ports are physically placed in the container.
- The flowitems and container can be separated by the separator object.
- The process time for the combiner starts as soon as all components are collected.

Join

- The flowitem from the first port is designated as the container.
- Flowitems are collected from other ports and then destroyed.
- The flowitem from Port 1 exits the combiner after the specified number of flowitems is collected and the process time is completed.
- A flowitem that leaves under the Join option cannot be separated.

Batch

- Once a flowitem enters through Port 1, the collection of flowitems from other ports begins.
- When the specified number of flowitems is collected, the process time starts.
- After the process time is complete, all flowitems collected from all ports are released in a single stream.

The various options or modes for combining flowitems are illustrated in Figure 8.2.

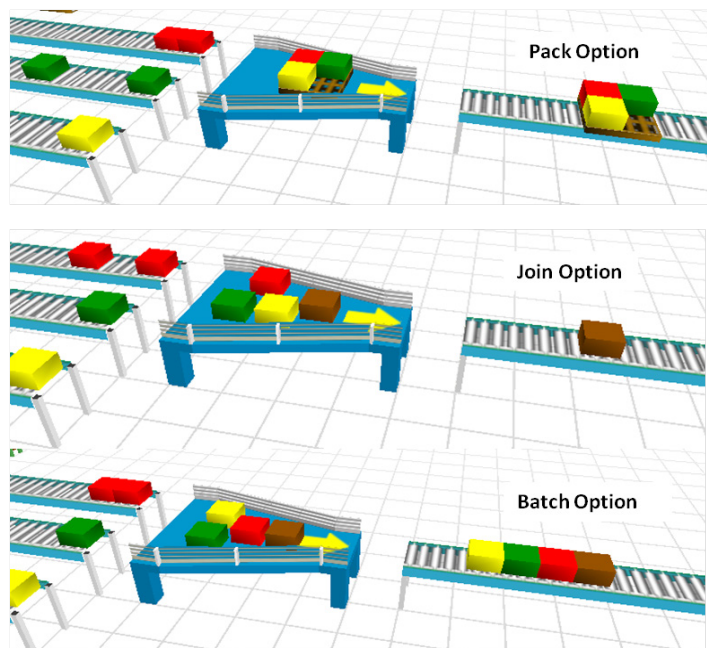


Figure 8.2 Combiner modes or options

Separator

The complement to the combiner is the separator object. It is primarily used to unpack flowitems that have been packed in a combiner. The separator can also be used to split or make copies of flowitems. The ProcessTimes tab for the separator contains picklist options that specify a Setup Time and a Processing Time. The processing time starts as soon as a flowitem enters the separator.

There are two separating modes that are available on the Separator tab of the object.

- Unpack: The container flowitem is separated from the other flowitems that make up the contents.
 - The default flow logic has the container exiting through the first port of the separator.
 - The Split Quantity has no impact when in Unpack mode.
- Split: The flowitem that enters the separator is split (duplicated) based on the picklist menu entry on the Separator tab.

The two separator modes are illustrated in Figure 8.3.

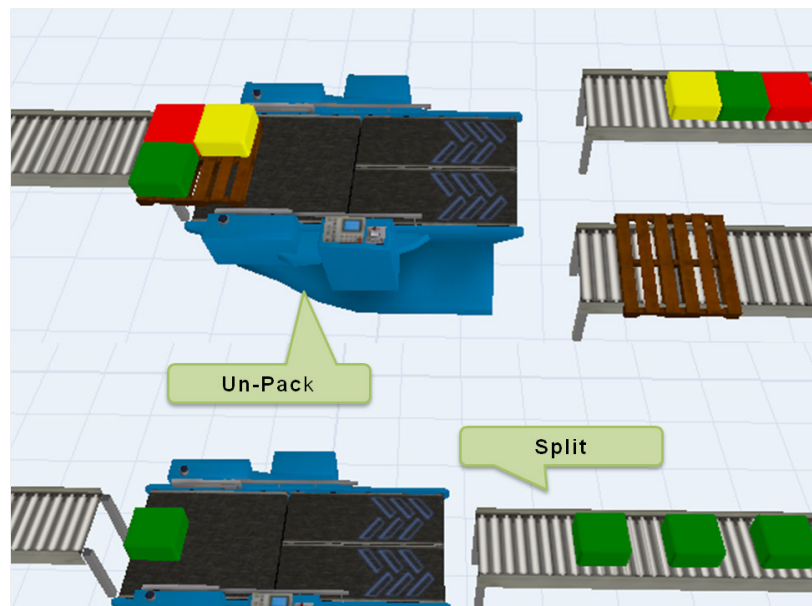


Figure 8.3 Separator options

Exercise 8-1 Hampton International

Background

In an effort to become the most advanced airport in the world, Hampton International decided to automate its luggage handling facilities. After lengthy delays, the system was put into operation but was immediately unable to provide the anticipated throughput. Passengers still had to wait for a long time for their luggage, and some

aircraft were kept waiting. The problem appears to be in the transportation of luggage from the aircraft to the distribution network.

Once in the distribution system, the bags are conveyed according to information on the luggage tags. The design consultant did not believe in simulating the system during the design phase and has subsequently been fired. The facility manager wants to simulate the operation to identify the problems.

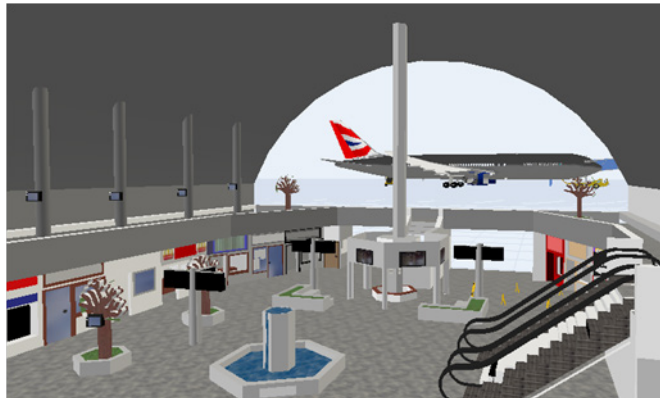


Figure 8.4 Hampton International

Problem statement

Prepare and use a simulation of the luggage transfer system from the aircraft to the distribution center to identify problems in the system, and test options to remedy the problems. Explore the 8-hour period of high volume.

Operating Data

As shown in Figure 8.5, under the new system, luggage is brought from the aircraft to a loading station where a series of transporters convey the luggage to the distribution point. Before reaching the distribution station, each transporter is inspected by Homeland Security. Once at the distribution point, the luggage is sent to its destination while the transporter is returned to the loading station.

The basic operating parameters for the system are as follows:

- On the average, luggage arrives at the transporter loading area from the aircraft about every 12 seconds (distributed exponentially).
- The loading area has space enough for 100 bags to wait to be loaded onto transporters.
- Eight pieces of luggage are put on a transporter. It takes about 20 seconds to fill the transporter as soon as all the pieces of luggage are present.
- The transporter brings the luggage to the inspection station; it covers the distance of 1200 feet in 1.5 minutes.
- The inspection station has 4 inspectors. Inspecting a transporter takes an average of 120 seconds; assume the times are exponentially distributed.
- After inspection, the transporter takes 1 minute to move the bags to the unloading station.

- The unloading process takes 45 seconds after which the transporter returns 1500 feet to the loading station travelling at a speed of 12.5 feet/second.
- Transporters enter the luggage loop at the start of each day on an entry track from their storage facility.

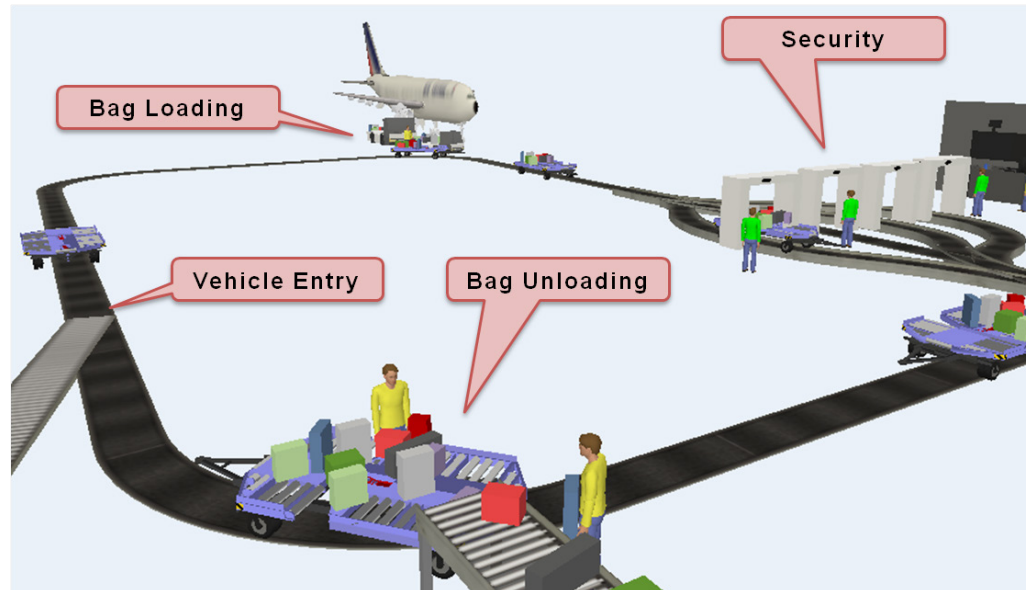


Figure 8.5 Hampton International baggage system

Expected results

- Develop an OFD for the system.
- How often are bags backed up outside the loading area?
- What should be the expected capacity of the bag retrieval system, in delivered bags per minute?
- Recommend what can be done to improve operations (e.g., more transports, more inspectors, faster transfers).

Modeling and analysis issues

- How detailed do you need to make the simulation? Can you represent all inspection stations with one processor?
- What time unit best serves this simulation?
- Think of how to easily set the travel time on the conveyors.
- What flowitem can be used to represent the transporters?
- How can you load a set of transporters into the model at the start?

- What combiner mode should be used to load bags on the transporter?
- How should the output ports of the separator be connected?
- What command can you use to give the bags more visibility?

Simulation details are included in the Appendix.

Section 8-2 Mobile resource objects (task executors)

In addition to the objects previously described to move flowitems, simulations often require mobile resources to move or carry items. These resources can include forklifts, automated guided vehicles (AGVs), or operators. Simulations may also require mobile resources to carry out certain tasks such as setting up or repairing a piece of equipment. The mobile resource objects, also referred to as task executors in *FlexSim*, can perform various functions:

- Physically move over the simulation surface
- Carry or transport flowitems between objects
- Execute a series of tasks
- Be assigned to an object in association with a particular operation (e.g., process setup, processing, maintenance)
- Distribute tasks to other task executors
- Follow a path created by network nodes

As shown in Figure 8.6, *FlexSim* offers a variety of types of task executors in the standard Discrete Objects library. In terms of their basic functionality, the objects are very similar. Note that there is a task executor object within the general class of task executors.

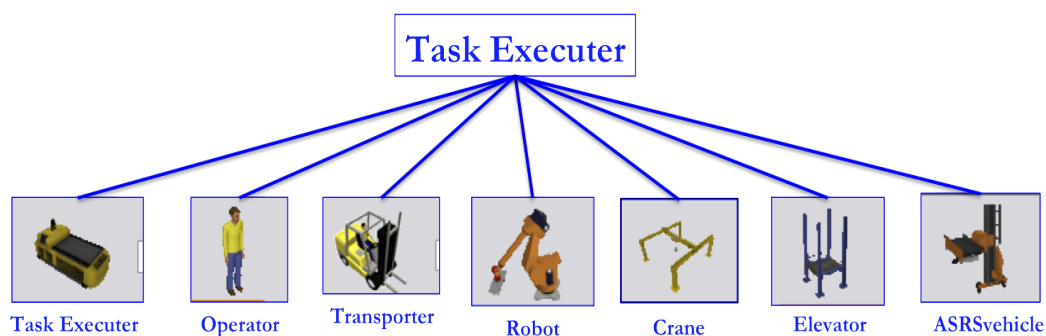


Figure 8.6 Type of task exectures used in *FlexSim*

The task executor object and operator objects are very similar. The main difference is that the operator can be animated to represent walking and carrying an item. The transporter object, analogous to a fork truck, can raise flowitems vertically using its

forks, as well as travel the surface. An elevator object only moves flowitems vertically, or in the Z direction. A robot moves flowitems between two points on the model surface from a fixed position using its arm (the base of the robot object does not actually move). Cranes and ASRS vehicles are more complex task executors. Most beginners use the basic task executors such as operators and transporters; however, it is helpful to be aware of what is available for more complex modeling projects. All task executors can travel along the model in three dimensions if attached to a path network.

As their name suggests, the primary function of task executor objects is to execute task sequences. A task sequence contains a series of specified tasks that are executed in sequence, has a priority value that indicates its importance to other task sequences, and can preempt or halt a task sequence that is being performed. Task sequences can be created automatically (e.g., when a fixed resource “uses transport”). In the case of Use Transport, the following tasks are executed:

1. Travel to the object that contains the item to be moved
2. Load the item from the calling fixed object to the task executor
3. Break; that is, check to see if there is some other task waiting (e.g., load another available item if the task executor has capacity)
4. Travel to the destination object
5. Unload the item from the task executor to the receiving fixed object

Another automatically generated task sequence occurs when assistance is needed to perform a process. In that case the task sequence is

1. Travel to the object that needs assistance with a process;
2. Utilize the task executor for the duration of the process.

One powerful feature of *FlexSim* is that the modeler can construct his or her own custom task sequences. This is an advanced topic; for more information, refer to the *FlexSim* User Manual. Task executor objects execute a task sequence if there is currently no active task sequence or if the incoming task sequence has a higher priority. Otherwise, it is dispatched to an associated task executor or it is queued for execution until the current task sequence is completed.

As shown in Figure 8.7, the main interface window for task executors is very different from other objects. The one shown is for a transporter type of task executor. All task executor objects have two main sections, one pertaining to operating properties of the object and the other addressing task executor behavior.

The first section of the interface is used to set values concerning the operation of the task executor; they vary somewhat depending on the role that the task executor performs.

Built-in attributes for task executors when being used to convey flowitems are as follows:

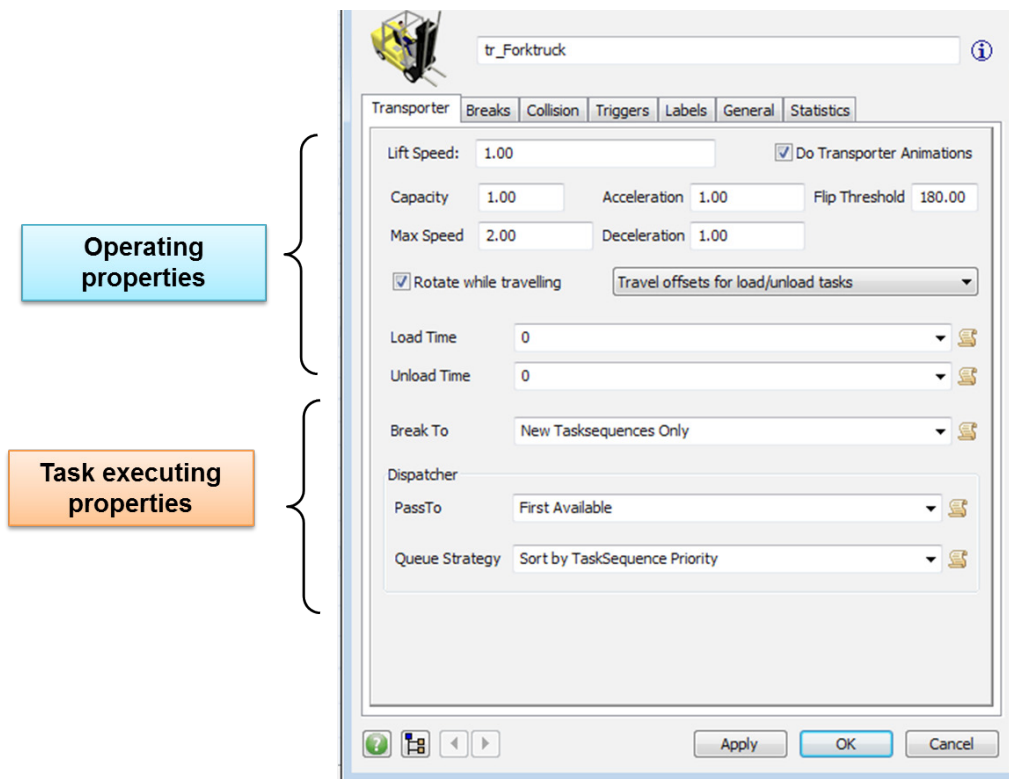


Figure 8.7 Interface for transporter object

- *Capacity*: This indicates the maximum number of flowitems that can be carried or transported at a time.
- *Load/Unload time*: This attribute gives the time, in simulation units, to load a flowitem at the originating object and the time to unload it at the destination object.
- *Lift Speed*: This indicates the speed in grid units per time unit.
- *Travel offsets for load/unload tasks*: If this option is not selected, the task executor will go to the origin of each object to load or unload flowitems. If selected, the task executor will go to the origin first and then move to the exact point where the flowitem is to be loaded or unloaded.
- *Flip Threshold*: When the angle between the transporter and the destination meets or exceeds this value, the transporter will “flip” to a mirror image in order to be facing the correct direction. This is for advanced visualization effects only.

Attributes when being used for any task are as follows:

- *Speeds (Max speed, Acceleration, Deceleration)*: The travel speed is in grid units per time unit and grid units per time unit squared, respectively

- The acceleration controls how fast a task executor reaches its maximum speed or how fast it slows down when it reaches its destination. Unless a very precise simulation is being conducted, it is advised to set the accelerations to 0 so that the task executor immediately achieves maximum speed or immediately stops at the destination. Alternatively, the acceleration and deceleration can be set to about one-half of the speed; this is, again, so that the object doesn't spend a lot of time accelerating or decelerating.
- The speed should be set to a reasonable value. For example, the typical walking speed of humans is 3 miles per hour (264 feet per minute, 4.4 feet per second, 75 meters per minute, etc.). Similarly, fork trucks travel 6 to 15 mph loaded and about 12 mph unloaded. Of course, these values should reflect the real systems operating environment and the grid size of the model surface and time unit of the model.
- Example: Assume a task executor has a maximum speed of 250 feet per minute and starts and ends at rest (speed = 0). Consider accelerations of 0, 75, 125, and 250 ft./min.². Using the basic laws of uniformly accelerated linear motion,

$v = a\Delta t$ where v = speed at the end of Δt , a = acceleration; and

$d = \frac{1}{2} a(\Delta t)^2$ where d = distance traveled.

The following table shows the effect of different acceleration/deceleration values on the time, in minutes, for the task executor to travel two distances, 100 feet and 1,000 feet. Note: setting the acceleration/deceleration to 0 really means infinite acceleration/deceleration; that is, the object instantly achieves maximum speed from rest or instantly comes to rest from maximum speed.

acceleration/ deceleration	d=100 ft.	d=1,000 ft.
0 (∞)	0.40	4.00
75	1.63	7.33
125	1.26	6.00
250	0.89	5.00

- *Rotate while Traveling*: If checked, the transporter will rotate as needed in order to orient itself in the direction of travel. This is for visualization effects only.

The second section of the interface deals with task behavior and involves advanced topics. Its options are mentioned here, but are not explained (consult the *FlexSim* Users Manual for more information).

- Break To: Logic that describes tasks that can interrupt the current task
- Dispatcher Pass to: Strategy for passing tasks along to other task executors. If more tasks are required than available task executors, the tasks are queued up.
- Queue Strategy: Logic for prioritizing tasks in the object's task sequence queue.

Task executors can share tasks; for example, operators can be assigned to respond to the needs of multiple pieces of equipment. To share tasks, the executors can be connected together. They can also be connected to a dispatcher object that assigns tasks based on priorities or other logic. Details for calling and connecting task executors can be found in the Appendix.

Network nodes

In Chapter 6, flow nodes were introduced as a means of visualizing the movement of flowitems from one object to another without the use of a piece of equipment such as a conveyor. Task executors often are required to travel complex paths, such as the paths that may be found when fork lift trucks travel among rows of a warehouse. In *FlexSim*, network node objects are used to provide such specific paths for task executors to follow as they move around the model surface. The primary interface tab for a network node is shown in Figure 8.8.

Establishing a network for task executors consists of three steps:

1. Connect network nodes to each other to form travel paths.
2. Connect network nodes to the objects that request services from task executors.
3. Connect task executors to the network.

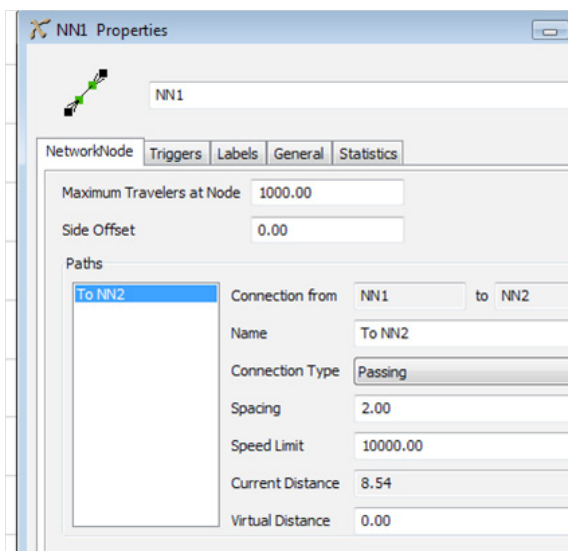


Figure 8.8 Network node interface and properties

Details of connecting the network nodes are contained in the Appendix.

The time required for a task executor to travel between two network nodes is the distance between the nodes divided by the speed of the task executor (assuming no acceleration or deceleration). The distance between the nodes is given in terms of the grid units. In Figure 8.8 network nodes 1 and 2 (NN1 and NN2) are 8.54 grid units apart. If the task executor travels at a speed of 20 grid units per minute, then the

task executer will take 0.427 minutes to travel between the two points. A convenient feature of network nodes in *FlexSim* is that a virtual distance between nodes can be specified. For example, if the virtual distance from NN1 to NN2 is set to 100 (grid units), then the task executer would take 5 minutes to traverse the path between the nodes ($100 \text{ grid units} \div 20 \text{ grid units per minute}$). Since network nodes are bidirectional, virtual distances need to be specified in both directions, from NN1 to NN2 and from NN2 to NN1.

Section 8-3 Displaying information on the screen

While all the objects and flow items add visualization to a simulation, there is often the need to add more content to the screen. In *FlexSim*, adding screen elements is accomplished by the visual tool object and the dashboard. These objects perform no direct actions in the simulation but provide a means of reporting information and decorating the model space.

Visual Tool

The main window for the visual tool is used for configuration. Possible representations, selected using the Visual Display picklist menu found on the Display tab, include the following:

- *Plane, Cube, Column, Sphere*: Basic shapes that can be decorated with imported picture files as texture
- *Imported Shape*: A 3D graphic that is added to the simulation surface.
- *Text*: Displays a single line of text including
 - basic information such as text or simulation clock time;
 - information obtained from an object connected to the visual tool by a center port connection (illustrated in Figure 8.9)
- *Presentation Slide*: Displays multiple lines of text

The visual tool in Figure 8.9 displays the output of a source, thus denoting the number of items entering the model. Information is passed from the source to the visual tool via a center port connection.

Visual characteristics such as size and color are modified on the General tab of the visual tool.

Visual tools can also act as a container for other objects in the model. When used as a container, the visual tool becomes a method for hierarchically organizing a model. The container can be saved in a User Library and act as a basic building block in the development of future models.

Visual tools provide an easy way to display information on the simulation surface.

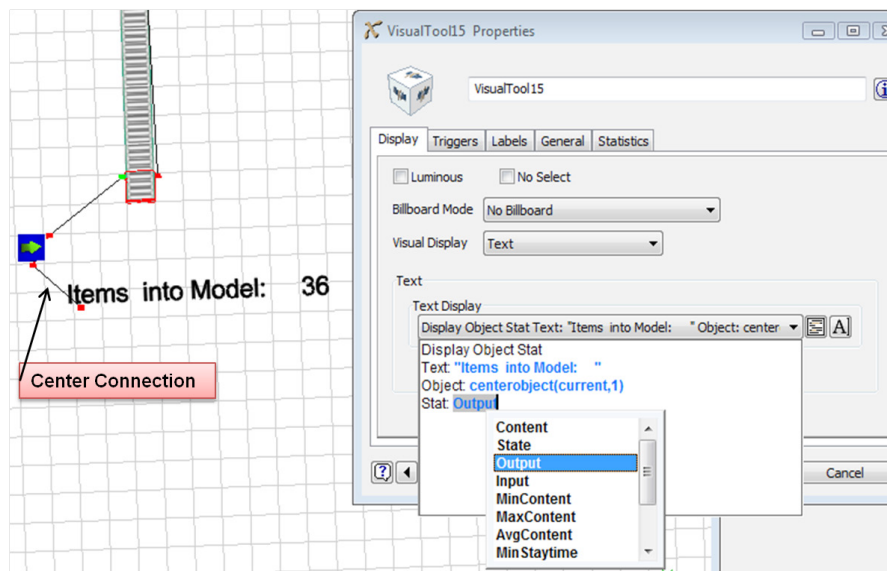


Figure 8.9 Visual Tool set to display the output of a source

Dashboard

The dashboard object allows dynamic model information to be grouped and displayed. The dashboard is created by selecting the menu option Statistics > Dashboard. Once opened, various text and graph displays can be added by choosing from the menus at the top of the dashboard. The object of interest is selected using the Universal Selection dialog that automatically opens. This dialog shows all the objects in the model organized by object type. More than one object can be selected to allow for comparisons. All the displays are updated as the model runs.

The displays can be moved and resized within the dashboard. Additionally, multiple dashboards can be created and organized by adding tabs using the icon in the upper right corner of the dashboard. Figure 8.10 shows a dashboard displaying dynamic information about the processor in a model.

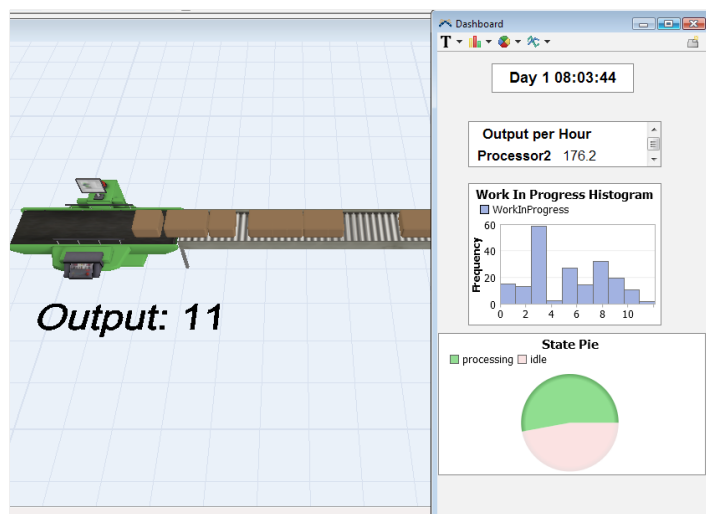


Figure 8.10 Dynamic information about model

Section 8-4 Establishing time tables

Time tables can be used to control an object during specific times in the simulation.

During the course of a simulation, many objects may start or stop, depending on the time of day. For example, workers may not be available during break times, or equipment may only be used on certain days. A common occurrence is stopping or starting equipment based on a particular work schedule. Often, custom programming is required to create and manage work schedules. In *FlexSim*, such time tables are established by building a time table in the Tools tab located on the main menu bar at the top of the screen.

Each time table can control many objects and each object can be controlled by multiple time tables. There are two independent ways to set up the table. The two methods are not synchronized, and only one should be used for any one table.

For simple occurrences, manually filling out the time table on the main window is best.

Consider an example of a small job shop that operates 12 hours (720 minutes) a day. At the end of the day everyone leaves, and then on the next day they pick up where they left off. There are three break times when no orders are accepted and all work stops:

- A 30 minute break after 3 hours (180 minutes)
- A one hour break after 5 hours (300 minutes)
- A 30 minute break after 8 hours (480 minutes)

In this case the three occurrences are independent and can be easily established using manual entries in a table as shown in Figure 8.11. The order processor is selected for the Time Table using the Universal Selection dialog.

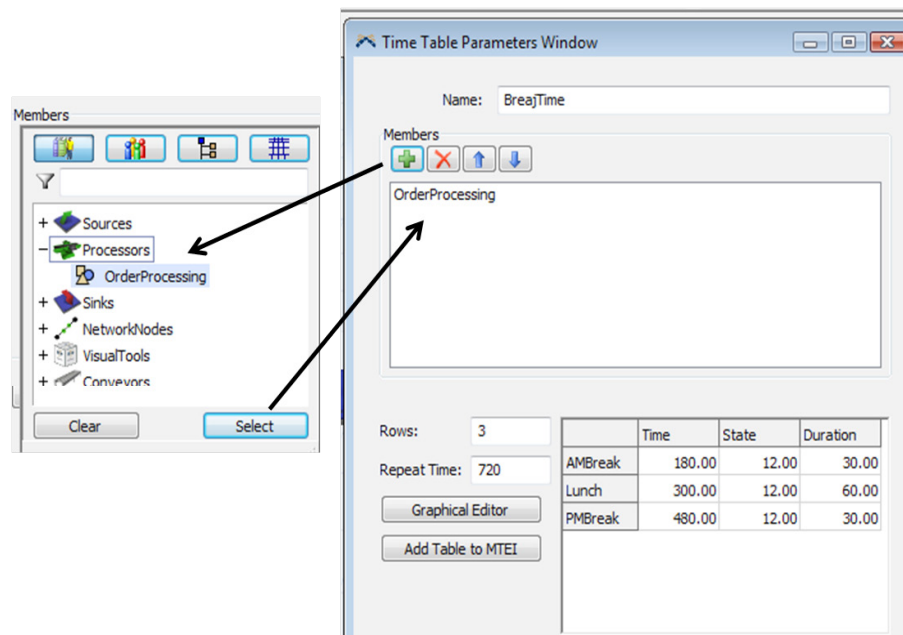


Figure 8.11 Time table for planned resource downtime

For more complex schedules, a graphical editor is available. These schedules change over the course of the simulation. Consider the above example, with an operation schedule that is open from 8 a.m. to 8 p.m. four days a week and a half day of Friday. This schedule would require the graphical editor.

Detailed configuration of the time table for each case is included in the Appendix

Exercise 8-2 Steve's Stone Cutting

Background

Steve Alpert transformed a large covered area, as shown in Figure 8.12, into a custom stone cutting operation. In the shop, large stones are cut to a specific size and a basic shape for various landscaping uses. As business grew, Steve gave priority to his wholesale customers.



Figure 8.12 Steve's Stone Cutting

Now that the retail business is nearly as large as the wholesale side, he's worried that by giving certain customers priority he will start losing new customers.

Problem statement

Provide a simulation that shows the current wait times for orders and the possibility for improving production for both types of customers while continuing to give priority to the wholesale market.

Operating data

Retail and wholesale jobs each arrive at the shop with an interarrival time that is exponentially distributed with a mean of 60 minutes. Steve has enough rough stones to fill all orders

The distances (feet) between the equipment and the entry and exit bays are:

From	To	Distance (ft)
Entry Bays	Rough Cutting	200
Rough Cutting	Finish Cutting	400
Finish Cutting	Exit Bay	200
Exit Bay	Entry Bay	400

The rough stones first go through a rough cutting process and then are taken for final cutting. The stones are transported by a fork lift truck between the entry bays (one for retail, one for product bay).

Process times in minutes to cut the stones are

- Rough Cutting – Triangular distribution: min. 10, max. 18, mode 12;
- Finish Cutting – Triangular distribution: min. 12, max. 20, mode 15.

The forklift travels at a speed of 50 ft per minute and travels around the work area on a path that is just inside all the equipment and bays.

Expected results

- Create an OFD for the system.
- Determine the number of each type of order that is received and processed during a 40-hour period.
- Note the average wait time for each type of order.
- What is the bottleneck?
- Estimate the benefits of reducing the travel time for the truck by clearing a path through the center of the shop area or having another forklift.

Modeling and analysis issues

- How will wholesale orders get priority? Is there an easy way in the simulation to change from priority to FIFO?
- Are network nodes needed? If so, how should they be set up to represent the correct distances?
- What visual changes can you make to better observe the material flow?
- Consider how to stockpile orders and send them to their respective shipping location.

Exercise 8-3 The Crafty Framer

Background

Your cousin has a great idea to sell customized picture frames and wants to get set up in time for the Christmas holiday season. The store is in a small location where customers can choose picture frames in all sizes and styles; however, the most profit will be made if customers choose to have the frame customized with specific decorations.



Figure 8.13 Crafty Framer

The customization process is straightforward and doesn't take much time, and the higher price can mean a greater profit margin. Since you mentioned having some knowledge of simulation, your cousin has asked you to simulate the shop operation and help decide how to best utilize the workers in the store. He currently has one checkout person and one custom framer but will hire another worker for the holiday rush.

Problem statement

Simulate the frame shop during a ten-hour period in order to help decide how to best utilize the three workers in the store.

Operating data

The frame shop operates from 9 a.m. to 7 p.m. At 7 p.m. the front door to the store locks. Customers in the store are serviced and the store is cleaned until 9 p.m. The procedure is repeated every day.

As shown in Figure 8.13, the store has two service counters—one used for customizing frames and one used as a check out area. The three workers can handle

either the check out or customizing station. During the holiday shopping period, customers are assumed to arrive at the shop at an average rate of 56 per hour; assume the times between arrivals are exponentially distributed.

There are three types of customers who enter at different rates:

1. Lookers, who just come inside without making a purchase, arrive at an average rate of 22/hr.
2. Base shoppers, who buy a frame without customization, arrive at an average rate of 20/hr.
3. Custom shoppers, who have their frames customized, arrive at an average rate of 14/hr.

Because of the size of the store and the local fire code, no more than 20 shoppers can be shopping at the same time; this does not count those in the checkout or custom lines.

Normally, customers spend about 15 minutes looking around at the merchandise before they either decide to leave, purchase a plain frame, or have one customized. The customization takes about 5 minutes while the checkout can be accomplished in about 2 minutes. Assume all times are exponentially distributed.

Retail marketing data show that customers who see more than 4 people in the customization line will just purchase the plain frames instead. If more than 10 people are in the checkout line the customers are likely to put their purchases down and walk out of the store.

Expected results

- Create an OFD for the system.
- Simulate 5 days of operation.
- Where should additional workers be used—cashier or custom desk?
- How many people were turned away because the store was full?
- How many settled for a standard frame because the custom line was too long?

Modeling and analysis issues

- Where can the demperical command be used?
- How can you increase the number of people on the checkout or custom counter without having to add additional objects?
- What activities take place in the store?

- What logic can be used to get people to the right place after they are done shopping? After finishing shopping what decision will a customer make?
- Consider how an object such as a queue, even though not a physical part of the simulation, can be used as a decision device to direct the flow using standard logic.
- How can you stop the shoppers coming in when the store closes but keep the simulation running until everyone leaves? Consider what to use as a starting time in the simulation.
- How long should you run the simulation to represent 5 operating days?

Chapter 8 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Explain the different functions that a combiner object can perform. Provide three operations system examples that would be modeled by a combiner object.
3. Describe the function of the flowitem that enters through the first input port of a combiner object.
4. Assume you want to batch 6 flowitems together using a combiner and that flowitems enter through two input ports. How many flowitems from each port will be grouped to form a batch?
5. Explain the different functions that a separator object can perform. Provide three operations system examples that would be modeled by a Separator object.
6. In the Unpack option of the separator, in what order do flow items leave the object?
7. The combiner and separator objects have processing capabilities similar to the processor object. For each object, describe at what point in time the processing activity begins.
8. Explain some of the actions performed by mobile resource objects.
9. What connection has to be made between a transporter, such as a fork truck, and the object that is calling it to perform a task? What tab on the calling object is used to call the transporter?
10. Assume a task executor is used to transport a flowitem between two objects. Complete the list below of basic or default tasks that the mobile resource would perform.

- 1) Travel to calling object.
 - 2)
 - 3)
 - 4)
11. Give an illustration of an operations system where a path network might be used.
 12. Identify four functions that can be performed with the Visual Tool object and give an example of each.
 13. Provide two examples of where a time table would be useful in a simulation other than the one given in the chapter.

Chapter

9

Modeling Randomness

Recall that a model is a representation of a system. The model is built to understand the behavior of the system and to help assess the consequences of actions on, or by, that system. A key aspect of model development is capturing the essence of the underlying structure and logic that exists in the system. Since discrete-event simulation models are stochastic (some of the inputs are probabilistic), another key aspect of model development is discerning the nature of the probability distributions used in the model. Many of the values used in a discrete-event simulation—times between arrivals, service times, quality indicators, time between failures, etc.—are obtained by taking random samples from probability distributions.

This chapter focuses on methodologies that are used to help decide what probability distributions to use in a model. This decision is either based on sample data obtained from the system, or it is assumed in the absence of data; both of these approaches are discussed in this chapter. The chapter also discusses how simulation software generates random samples from probability distributions—this includes discussing both generating random variates and generating random numbers. The chapter also discusses how random samples drive events and the simulation itself.

Section 9-1 Data-driven probability distribution selection

Oftentimes distributions that are used in simulation models are based on data obtained from the system being modeled. This involves taking samples from the random processes (e.g., service times or times between arrivals), “fitting” that data to standard or theoretical probability distributions (e.g., normal, exponential, lognormal, or Weibull), and inferring which distribution best represents the population in the real system. If the sample data does not fit a standard distribution well, then an empirical distribution is used; that is, one that closely approximates the probability

Simulating real systems means simulating randomness.

Where data is available, a statistical distribution to represent that data may be found.

distribution of the sample data. In both cases, a probability distribution is selected based on data sampled from the system being modeled.

Selecting distributions from sample data is based on the concept of goodness-of-fit testing from statistics; that is, sample data are compared to theoretical distributions such as the normal, exponential, etc., in order to infer the true population distribution. Goodness-of-fit testing is analogous to making a visual comparison on how well different theoretical distributions fit the sample data. Of course, goodness-of-fit testing is a more formal and rigorous approach since it is grounded in statistics.

In Figure 9.1, the histogram represents the sample data (in this case, 50 observations) collected from the system being modeled. Three theoretical distributions—Gamma, Erlang, and Lognormal—are overlaid in order to see which best represents the sample data. As the plot indicates, all are similar in their representation of the sample data.

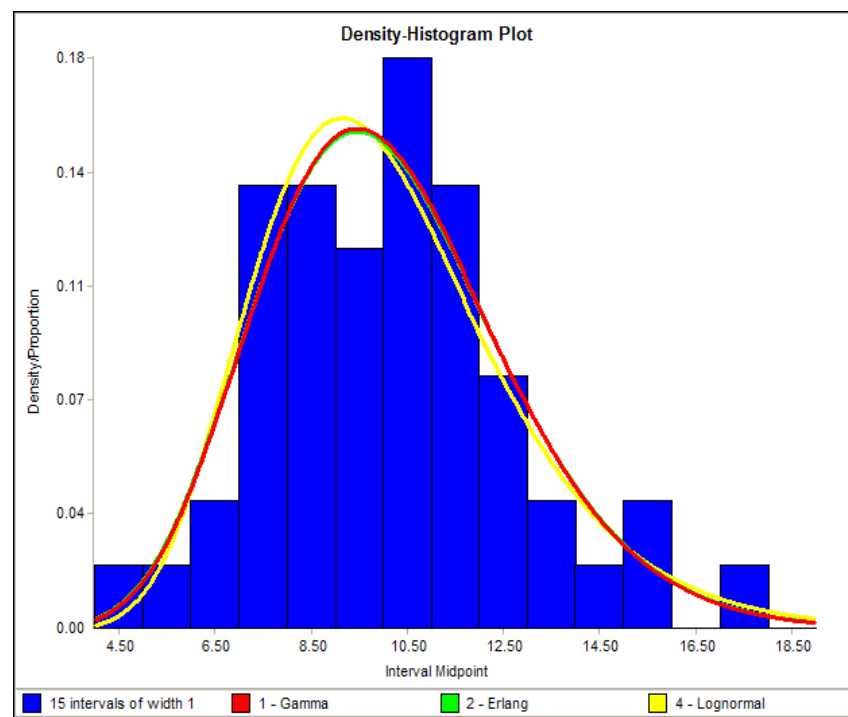


Figure 9.1 Comparison of sample data and theoretical distributions

It is interesting to note in this case that the example data were sampled from a normal distribution; that is, the true, or population, distribution is normal. Of course, in practice you would never know the true distribution; otherwise, there would be no need to take the sample and perform the tests. In fact, the data were generated by the normal function in *Excel*. The point is that the distribution selection, like any other statistical analysis, is based on sample data and the inference may or may not be correct. That is the risk that one takes by not examining the entire population, a task that most of the time is impractical or impossible.

Discrete data can similarly be analyzed. Consider the example in Figure 9.2. In this case the data, shown in blue (dark-colored bar) in the histogram, are the frequency of the number of items customers purchase (about 16% of the customers buy 1 item, 18% 2 items, etc.). The data were obtained from a recent study of a retail store's operations. In Figure 9.2 the sample data are compared to the discrete uniform theoretical distribution, which provides the best fit according to the statistical tests. The discrete uniform frequencies are shown in the red, or the light-colored bars. However, in this case the analyst decides to use the sample data in the model and not the theoretical distribution. The data will be represented as a discrete empirical distribution. In *FlexSim*, the probability distribution would be implemented through the `dempirical()` command and the mass function of the sample data would be stored in a global table.

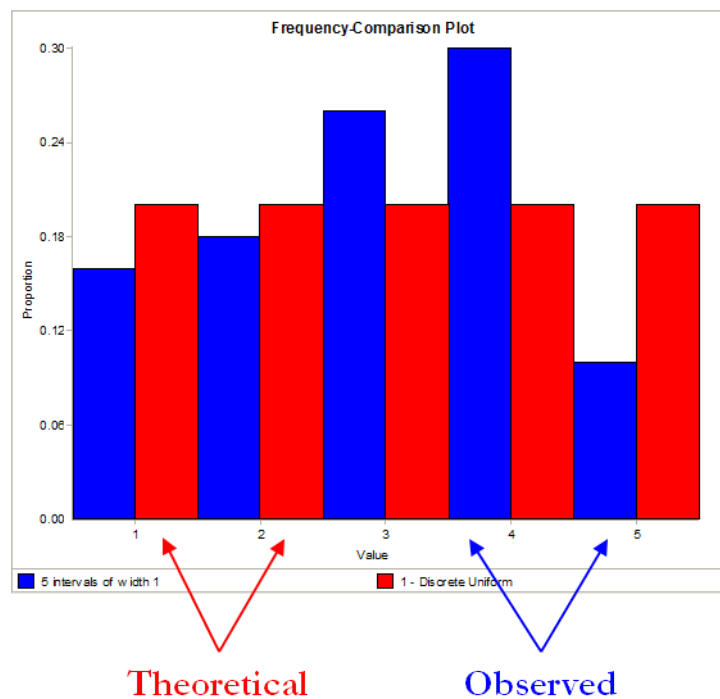


Figure 9.2 Goodness-of-fit testing with discrete data

It is common to use data from the system being modeled to estimate the probability distributions that are inputs to the model. Such input probability distributions specifications include inter-arrival time, service time, time between failures, repair time, quality, etc. Distribution fitting can also be used to estimate the distribution of output from the model, such as cycle time, inventory level, service rate (probability customers wait more than a specified threshold), etc. For key performance measures, distributions are much more informative than point or interval estimates.

There is software available to perform the statistical test and help in selecting, based on sample data, the best distribution to use in a simulation model. The software typically recognized as the leader in the market is *ExpertFit*; it is produced by Dr. Averill Law, one of the leading experts in the field of discrete-event simulation

modeling and analysis, and is included with *FlexSim* and other simulation packages. Figure 9.3 provides example input user interfaces from *ExpertFit*; they indicate the range of distributions that can be tested and the types of statistical tests used to guide the distribution selection process. Figure 9.1, which compares the sample data with several theoretical distributions, is an example output from *ExpertFit*.

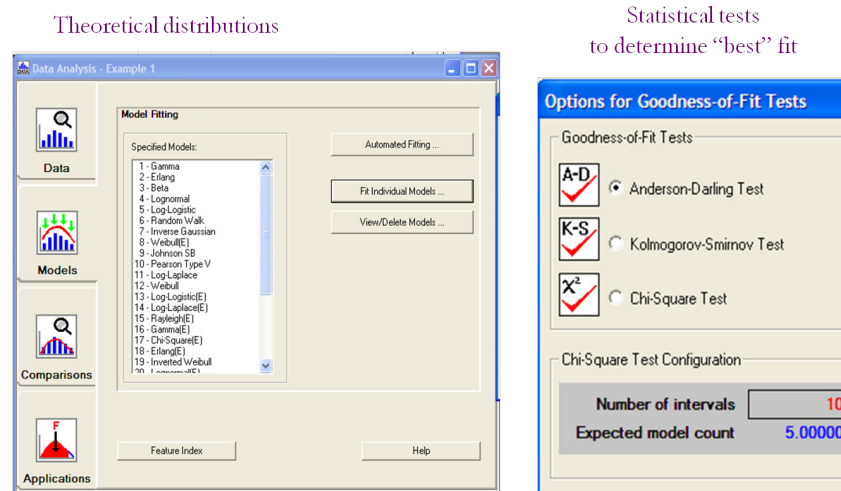


Figure 9.3 Distributions and statistical tests in *ExpertFit*

ExpertFit not only provides the means to analyze raw data; it also allows the user to view statistical distributions and to estimate distributions when only general observations are available. The application provides two modes of operation when determining what distribution best fits a data set: Standard Mode, which fits most cases, and an Advanced Mode, which contains additional features for an Advanced User. There are also two levels of precision for the data analysis. The *ExpertFit* application that is provided as part of the *FlexSim* software contains an extensive Help file as well as tutorials.

The appendix to this chapter contains information on key probability distributions used in simulation and more information on *ExpertFit*.

Section 9-2 Selecting probability distributions in the absence of data

There are situations when system data are not available; for example, the system may not yet exist. In those cases, engineering judgment based on knowledge of the proposed system or comparable systems may be used to subjectively specify the distribution.

Another approach is to use a generally flexible distribution with easy-to-estimate parameters. One such distribution that is commonly used to represent continuous random variables in the absence of data is the triangular distribution. The density function is shown in Figure 9.4. The parameters to specify the triangular distribution are the minimum, maximum, and most likely values. These are usually easier to

specify than the mean and standard deviation in the case of the normal. Depending on the values of the parameters, the triangular distribution can be skewed or symmetric.

System data is not always available so other methods can be used.

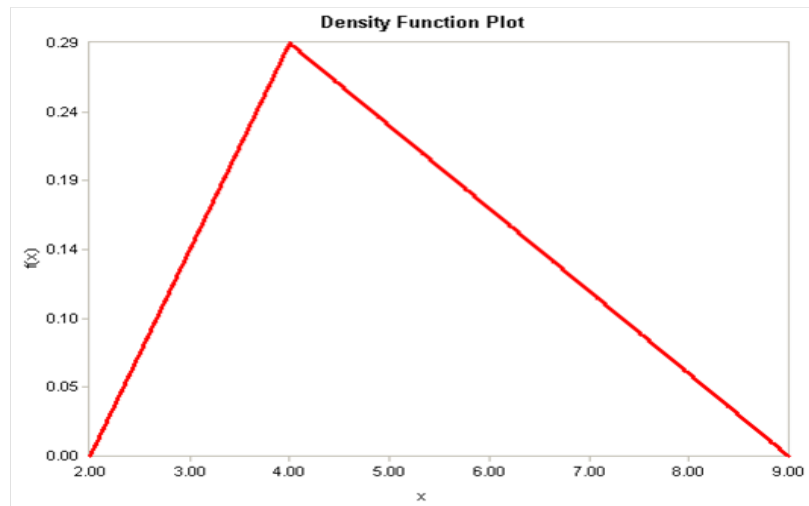


Figure 9.4 Triangular distribution

In the case of Figure 9.4, the minimum, maximum, and most likely values are 2, 9, and 4, respectively, assuming the distribution is of processing times. In this case, it is assumed that process times will not be less than 2 minutes or more than 9 minutes, with the mode or most likely value being 4 minutes. The mean or average process time is 5 minutes; in general, the mean μ of a triangular distribution is

$$\mu = \frac{\text{min} + \text{max} + \text{mode}}{3}.$$

Other distributions, such as the Weibull and lognormal, are used to model continuous distributions when data from the real system are not available. Since it is oftentimes difficult to specify the maximum value, those distributions allow the upper end of the distribution to be specified by percentiles rather than using an absolute maximum. However, *ExpertFit* provides a means for specifying a percentile of the triangular distribution instead of the maximum. For example, if the 90th percentile for triangular distribution shown in Figure 9.4 had been specified as 7, then the maximum would be 8.81 (close to the 9.0 maximum in Figure 9.4).

The three distributions – triangular, lognormal, and Weibull – are compared in Figure 9.5. All have the same minimum, mode, and 90th percentile values of 2, 4, and 7, respectively. All have similar means (4.94, 4.96, and 4.75, respectively) and similar standard deviations (2.04, 2.60, and 2.58, respectively). Of course, the triangular has an upper bound of 8.81. The lognormal and Weibull have infinite upper bounds; therefore, it is possible to obtain a large value in a random sample, although it is not very likely.

One downside to the lognormal and Weibull distributions is that their parameters (location, scale, and shape) are less intuitive to the layperson than those of the

triangular (minimum, maximum, most likely). Of course, using a tool such as *ExpertFit* allows the distribution to be specified in terms of minimum, most likely, and 90th percentile and then translated to the appropriate parameters. For example, a Weibull that has minimum, most likely, and 90th percentile values of 2.00, 4.00, and 8.81 is specified in terms of location, scale, and shape parameters as Weibull(2.00, 3.14, 1.80)

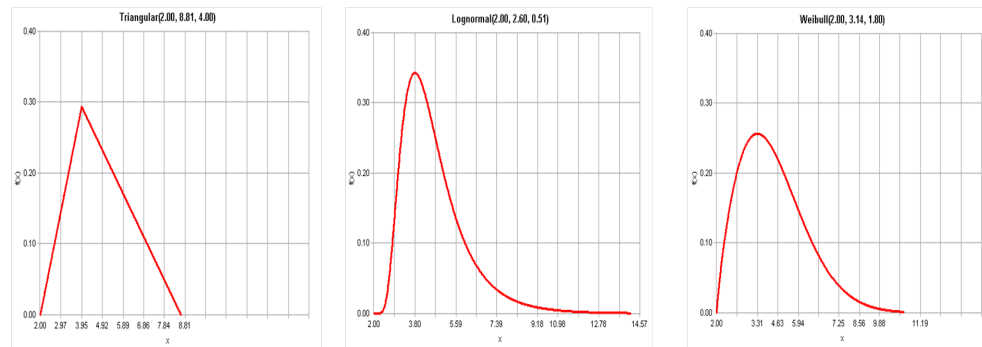


Figure 9.5 Comparison of similar distributions

Whenever data are not available to specify a probability distribution, it is advisable to perform sensitivity analyses on the uncertain distributions in order to identify which ones significantly impact the results from the simulation model. The distributions that have high impact need further scrutiny since they could affect the decisions that are being made based on the simulation model results. Not all inputs to a simulation model have the same impact on output; therefore, sensitivity analyses are used to identify the drivers (the parameters that are the most sensitive) and to direct further investigation into the value of the parameters for that variable. Time and resources are always scarce in a simulation project; therefore, efforts should be directed at performing better estimates only on those parts of the model that have the most impact on results.

Section 9-3 Use of probability distributions in simulation

Whether probability distributions are derived from sample data obtained from the system being studied or subjectively, they are a key driver in simulating system behavior. The remainder of this chapter describes how probability distributions are sampled from in a simulation and how they are used to drive events.

Figure 6.1 presented the basic components found in all professional discrete-event simulation software. That figure is presented again in Figure 9.6 but with four components highlighted – those that deal with generating random events. Sections 9-4 and 9-5 discuss the first component, generating random variates in terms of continuous and discrete distributions. Section 9-6 describes the second component, how random numbers are generated in a simulation. Section 9-7 describes the combination of the first two components that produces random samples from probability distributions. Section 9-8 describes how the random samples drive events and the

simulation clock. The chapter concludes with a discussion of a methodology for reducing variability in the samples.

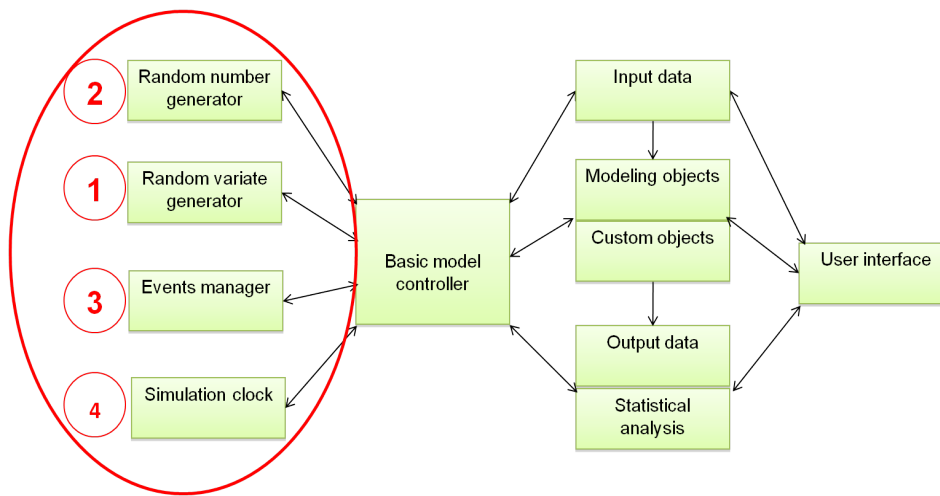


Figure 9.6 Simulation software components that drive events.

Section 9-4 Sampling from continuous random variates

Once probability distributions are specified in a simulation model, the software must repeatedly sample from those distributions as the model runs. In order to invoke randomness in a simulation, the model developer only has to specify the probability distributions and their parameters. It is, however, useful to know how the sampling occurs within the software. This section describes the process used by most simulation software to sample from continuous probability distributions; the next section describes the process for sampling from discrete probability distributions.

In general, consider $f(x)$ to be a continuous probability density function, and $F(x)$ is the cumulative distribution function where $0 \leq F(x) \leq 1$. In general, $F(x)$ is defined as:

$$F(x) = \int_{\text{lower limit}}^x f(w) dw .$$

As shown in Figure 9.7, $F(x)$ represents the probability that the random variable X will be less than the specified value x ; this is also the area under the $f(x)$ curve from the lower bound of $f(x)$ to x .

Also, consider u_i to be a random number uniformly distributed between 0 and 1. Since u_i and $F(x)$ have the same domain (they can take on values between 0 and 1), u_i is set equal to $F(x)$, $u_i \equiv F(x_i)$, and the equation is used to solve for x in terms of u_i ; that is, $x_i \equiv F^{-1}(u_i)$. This approach is referred to as the inverse transformation method. Two examples are provided to illustrate how the inverse transformation works—sampling from the uniform and exponential distributions.

In the simulation, samples are taken from specified distributions.

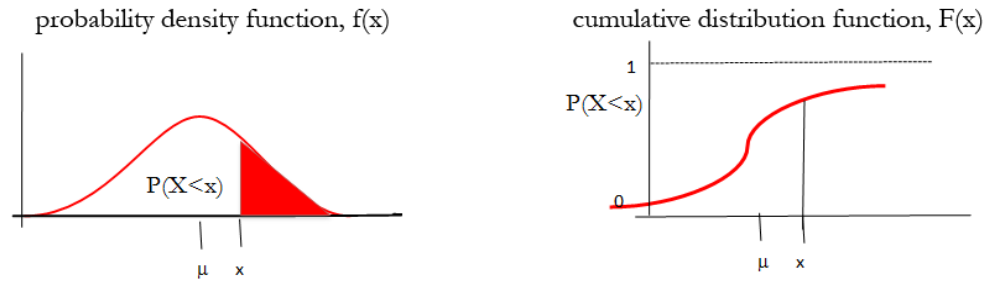


Figure 9.7 Probability density and distribution functions

The plots for probability density and cumulative distribution functions for the continuous uniform distribution are shown in Figure 9.8. The density implies that if the random variable X is uniformly distributed between α and β , then any value of X is equally likely to occur across the interval from α to β .

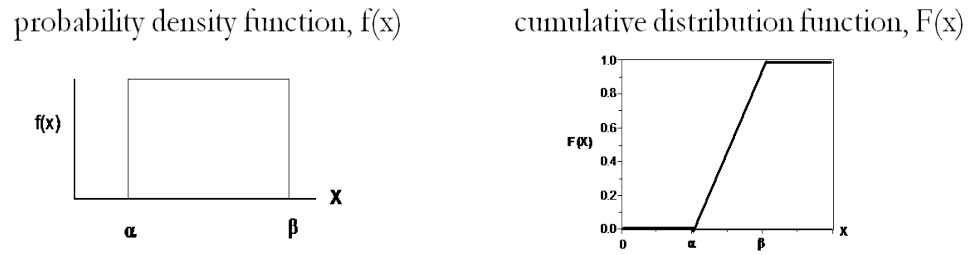


Figure 9.8 Uniform density and distribution functions

The density and distribution functions are

$$f(x) = \frac{1}{\beta - \alpha} \text{ for } \alpha \leq x \leq \beta,$$

$$F(x) = \int_{\text{lower limit}}^x f(w) dw = \int_{\alpha}^x \frac{1}{\beta - \alpha} dw = \frac{w}{\beta - \alpha} \Big|_{\alpha}^x = \frac{x - \alpha}{\beta - \alpha} \text{ for } \alpha \leq x < \beta,$$

$$F(x) = 0 \text{ for } x < \alpha \text{ and } F(x) = 1 \text{ for } x \geq \beta.$$

Since the density function can be integrated and is given by $F(x)$, the inverse transformation method can be applied. As shown below, $F(x)$ is set equal to a $U(0,1)$ value u_i , and the resulting equation is solved in terms of x_i , the sample value from a $U(\alpha, \beta)$:

$$F(x) \equiv u \Rightarrow \frac{x_i - \alpha}{\beta - \alpha} \equiv u_i \Rightarrow x_i = \alpha + (\beta - \alpha)u_i.$$

For example, assume travel times are uniformly distributed between 100 and 300 seconds; that is, $\alpha=100$ and $\beta=300$. In order to get a random sample from the $U(100, 300)$ distribution, a $U(0,1)$ random number is generated (more on how that

is done later), say $u_1=0.67306$, and applied to the equation above, as seen in the following:

$$x_1 = \alpha + (\beta - \alpha)u_1 = 100 + (300 - 100)0.67306 = 234.6.$$

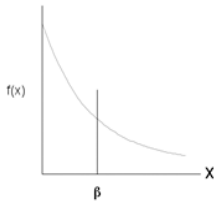
The inverse transformation method is also used to sample from the exponential distribution. The probability density and cumulative distribution functions for the exponential (actually the negative exponential) distribution are provided below. Plots of the functions are shown in Figure 9.9. The only parameter required to specify the exponential is its mean (β); that is, the density implies that the random variable X is exponentially distributed with mean (β), $X \sim \exp(\beta)$.

$$f(x) = \frac{1}{\beta} e^{-\frac{x}{\beta}} \text{ for } x \geq 0$$

$$F(x) = \int_{\text{lower limit}}^x f(w) dw = \int_0^x \frac{1}{\beta} e^{-\frac{w}{\beta}} dw = -\frac{1}{\beta} e^{-\frac{w}{\beta}} \Big|_0^x = 1 - e^{-\frac{x}{\beta}} \text{ for } x \geq 0$$

$$F(x) = 0 \text{ for } x < 0$$

probability density function, $f(x)$



cumulative distribution function, $F(x)$

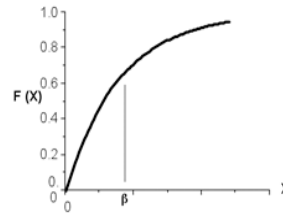


Figure 9.9 Exponential density and distribution functions

Since the density function can be integrated and is given by $F(x)$, the inverse transformation method is applied. As shown below, and as was done in the previous example, $F(x)$ is set equal to a $U(0,1)$ value u_i , and the resulting equation is solved in terms of x_i , the sample value from $\exp(\beta)$.

$$F(x) \equiv u \Rightarrow 1 - e^{-\frac{x_i}{\beta}} \equiv u_i \Rightarrow x_i = -\beta \ln(1 - u_i)$$

For example, assume travel times are exponentially distributed with a mean of 200 seconds; i.e., $\beta=200$. In order to get a random sample from the $\exp(200)$ distribution, a $U(0,1)$ random number is generated (more on how that is done later), say $u_1=0.67306$, and applied to the equation above:

$$x_1 = -\beta \ln(1 - u_1) = -200 \ln(1 - 0.67306) = 223.6.$$

Of course, the inverse transformation method can only be used if $F(x)$ exists. For many distributions—such as the uniform and exponential illustrated above—the density function can be integrated to obtain the cumulative distribution. This is not the case for other distributions, such as the normal; therefore, other methods must be used to sample from these distributions. The inverse transformation method is considered a direct method; other direct methods are the composition and convolution methods. The acceptance-rejection method is an example of an indirect method. For more information on these methods, see *Simulation Modeling and Analysis*, by Law.¹

The following list identifies the typical method used by simulation software to sample from some common probability distributions. Again, for more information, refer to Law's book.

- *Uniform*: Inverse transformation
- *Exponential*: Inverse transformation
- *Erlang*: Inverse transformation & composition
- *Gamma*: Multiple methods
- *Weibull*: Inverse transformation
- *Normal*: Multiple methods
- *Beta*: Acceptance-rejection
- *Lognormal*: Multiple methods
- *Triangular*: Inverse transformation

Section 9-5 Sampling from discrete random variates

The concept for sampling from discrete distributions is the same as that for continuous distribution as described in the previous section. The process is illustrated through an example for an empirical distribution. Assume the probability of the number of items purchased by customers in a local store is as follows. That is, 10% of the customers are expected to buy one item, 15% are expected to buy two items, 40% are expected to buy three items, etc.

x	p(x)
1	0.10
2	0.15
3	0.40
4	0.25
5	0.10

In order to sample from that distribution, the cumulative probability distribution is used to set up ranges for the random numbers. As shown in Figure 9.10, if a $U(0,1)$ random value is between 0.000 and 0.099, then the simulation will have that customer purchase one item. Similarly, if the $U(0,1)$ value is between 0.100 and 0.249, then the simulation will have the customer purchase two items. Using this approach, in the long run in the simulation, 10% of the customers will purchase one item, 15% will purchase two items (alternatively, 25% will purchase two or less items), etc.

X	p(x)	P(x)	RN
1	0.10	0.12	0.000 to 0.099
2	0.15	0.25	0.100 to 0.249
3	0.40	0.65	0.250 to 0.649
4	0.25	0.90	0.650 to 0.899
5	0.10	1.00	0.900 to 0.999

Figure 9.10 Sampling from discrete probability distributions

In a simulation, given a $U(0,1)$ random number (as discussed in the next section), the simulation generates values for the random variable via table lookup. For example, using the table above, if $u_1 = 0.67306$, then $x_1 = 4$. Similarly, if $u_2 = 0.25107$, then $x_2 = 3$ and if $u_3 = 0.86151$, then $x_3 = 4$.

Section 9-6 Generating random numbers

In the previous sections it was taken for granted that the simulation software could generate random numbers that are uniformly distributed between 0 and 1, $U(0,1)$. These random numbers are used to sample from probability distributions or generate random variates. This section describes how simulation software generates the random numbers.

One approach to generating random numbers is to include an extensive random number table in the simulation software. A small portion of a random number table is found in most introductory statistics books; however, accessing this table would be highly inefficient from a processing standpoint, given the large number of random numbers used in a simulation. The approach used in commercial simulation packages generates numbers that are not actually random, although they appear to be random. The numbers are generated by a recursive relationship, and since they are not really random, they are referred to as pseudo-random numbers; however, for brevity, for the remainder of the book we simply refer to pseudo-random numbers as random numbers.

A key part of simulation software is a random number generator. Generators produce numbers that are uniformly distributed between 0 and 1 and are reproducible. Having reproducible random numbers is important for debugging and verification, and it enhances comparisons. Differences are due to changes in the model or

Samples may also be taken from a discrete distribution for use in the simulation.

Generating random numbers is a common activity in the simulation but must be understood.

system and not attributed to randomness. The generators are fast and require a small amount of storage, and they are portable; that is, they can be used with different compilers and on different computers.

Most simulation software uses either multiplicative or mixed linear congruential generators (LCGs). The ones that are used in commercial simulation packages have been extensively tested. It is important that these generators provide independent values and have long periods. A *period* is the length of a sequence of values before they repeat. Since most simulations require millions of samples from distributions and thus millions of random numbers, a generator with a long period is essential. As will be seen in the example below, the generators are quite simple; however, finding good parameters is very challenging. Much research and testing has been done to identify good parameter values.

There are other types of generators besides LCDs (e.g., more general congruentials, composite, combined multiple recursive). For more information on these different types of generators and the types of statistical tests that are used to identify good generators see *Simulation Modeling and Analysis* by Law.¹

The following mixed LCG is a recursive relationship that produces a sequence of integers:

$$N_i = (aN_{i-1} + c)(\text{mod } m),$$

where a is the multiplier, c the increment, m the modulus, and N_0 the seed. All are non-negative integers that, as discussed above, must be set carefully. The quality of the generator is very sensitive to the choice of its parameter values. Note, we use the notation N_i , N_{i-1} , and N_0 in this text; however, many references use the notation Z_i , Z_{i-1} , and Z_0 . We change notation so as not to confuse this with the standard normal distribution notation.

Since the generator produces integer values, noted as N_i , and the simulation needs uniformly distributed values between 0 and 1, noted as $U(0,1)$, u_i is obtained by dividing the random number by the modulus m :

$$u_i = \frac{N_i}{m}.$$

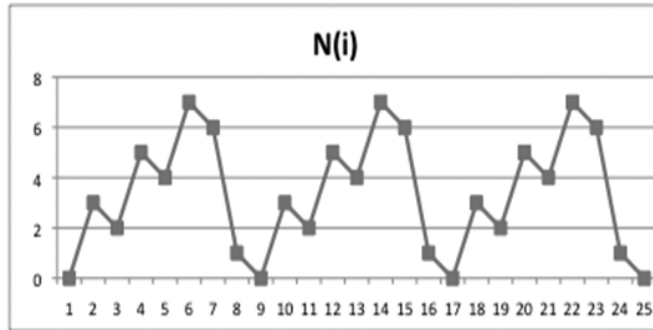
In order to illustrate how the random number generation process works, consider the example where $a=5$, $c=3$, $m=8$, and the $N_0=1$. That is,

$$N_i = (5N_{i-1} + 3)(\text{mod } 8).$$

While this is obviously a poor generator, it serves to illustrate the process. One problem with this generator is that its period will never be longer than m ; therefore, in this case, the maximum period is just eight numbers before they repeat. With the seed $N_0=1$, the generator produces the following sequence of random numbers: $N_1=0$, $N_2=3$, $N_3=2$, etc. The resulting sequence of $U(0,1)$ values is: 0.000, 0.375, 0.250, etc. The results are shown in Figure 9.11.

Not all generators produce good random number streams.

Notice that the generator produces all integers between 0 and 7 ($m-1$), the most integers possible with a modulo value of 8; therefore, it is considered full period, albeit very small. After generating those eight values, the pattern repeats. Again, the $U(0,1)$ values are obtained by dividing the random number from the generator by m .



i	N(i)	u(i)
0	1	
1	0	0.0000
2	3	0.3750
3	2	0.2500
4	5	0.6250
5	4	0.5000
6	7	0.8750
7	6	0.7500
8	1	0.1250
9	0	0.0000
10	3	0.3750
11	2	0.2500
12	5	0.6250
13	4	0.5000
14	7	0.8750
15	6	0.7500
16	1	0.1250
17	0	0.0000
18	3	0.3750
19	2	0.2500
20	5	0.6250
21	4	0.5000
22	7	0.8750
23	6	0.7500
24	1	0.1250
25	0	0.0000

Figure 9.11 Random numbers produced by a simple generator

Section 9-7 Putting it all together—obtaining samples from probability distributions

As indicated earlier and as should be apparent through the model building discussed so far in the text, probability distributions play a huge role in simulation modeling and analysis. The sections at the beginning of this chapter discussed ways to decide the appropriate probability distribution to use in a simulation model in order to best represent system behavior. The next section discusses how simulation software obtains samples from the specified distributions, in terms of generating random variates using random numbers. The following is a summary of that process.

1. Specify the distribution and its parameters that best represent the randomness in the system being modeled; that is, some element in the simulation model is a random variable, X .
 - a. Determine the distribution of X and its parameters; for example, $X \sim \text{normal}(\mu, \sigma)$ or $X \sim \text{exp}(\beta)$. Make this determination either through a statistical goodness-of-fit testing process using data from the system (typically with the help of software such as *ExpertFit*) or make the determination based on engineering judgment in the absence of data.
 - b. Use either a theoretical distribution (e.g., normal, exponential, lognormal) or an empirical distribution.

2. Each time a random value is needed in the simulation, do the following
 - a. Randomly generate a $U(0,1)$ number u_i using a pseudo-random number generator. For example, use a linear congruential generator such as $N_i = (aN_{i-1} + c) \pmod m$ and convert the random integer to a $U(0,1)$ random variable by $u_i = N_i/m$.
 - b. Use u_i to sample from the specified probability distribution. For example, apply the inverse transformation method and use the cumulative distribution function, $F(x)$; set $u_i = F(x_i)$ and solve for x , $x_i = F^{-1}(u_i)$. For an exponentially distributed random variable with a mean β , samples from the random variable X , where $X \sim \exp(\beta)$, are obtained by executing $x_i = \beta \ln(1-u_i)$.

Section 9-8 Using random samples to drive a simulation

In general, the random samples from probability distributions create the behavior of the simulation. Oftentimes, most of those samples create event times that change states in the system and thus drive the dynamics of the simulation. This is illustrated in Figure 9.12 through a simple, single-server queueing system. In this case, the state is the number of customers currently in the queue. It is assumed that simulation starts with the queue empty but the server busy; customers are processed in a first-in first-out (FIFO) manner.

In Figure 9.12, a series of events – customer arrivals and departures – occur over time and change the state of the system. An arrival event, denoted as a red upward-pointing triangle, increases the number in queue and a departure event, denoted as a green downward-pointing triangle, decreases the number in the queue. These events correspond to randomly-generated inter-arrival times (time between arrivals) and service times. The state of the system does not change between events.

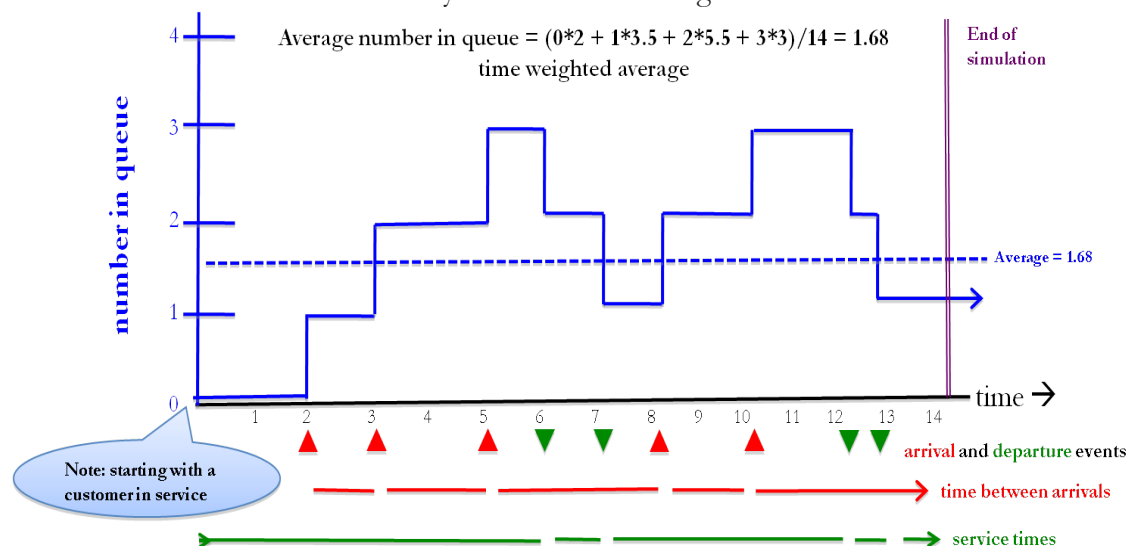


Figure 9.12 Random samples create state changes and produce dynamic behavior

A third event type, shown in Figure 9.12, is the end of simulation event. In the figure the simulation is scheduled to run 14 minutes. In practice, this simulation time would be much too short for analysis; however, it is sufficient in this example to illustrate the event processing concept.

States of the system are often summarized and used as performance measures. In Figure 9.12, the average number in queue is a measure of interest. Over the 14 minutes of the simulation, as shown below, the average number in the queue is 1.68 customers. This is the time-weighted average of how long the system was in each of the possible states. That is, there were 2 minutes when there were 0 in queue (state = 0), 3.5 minutes when there was 1 in the queue (state = 1), etc.

$$\bar{L}_q = \frac{\sum_{i=0}^n i * t_i}{\sum_{i=0}^n t_i} = \frac{(0 * 2 + 1 * 3.5 + 2 * 5.5 + 3 * 3)}{14} = 1.68$$

The logic that is followed for a simple, single-server FIFO system is shown in flowchart format in Figure 9.13. As in Figure 9.12 there are only two types of events, other than the end-of-simulation event, arrivals and departures. As shown in the figure, if the current event is an arrival, then the first action is to create and schedule the next event. This typically involves adding the value obtained from a random sample from the inter-arrival time distribution to the current clock time and placing that time in an events list. The remaining actions for the arrival event depend on whether or not the server is busy. If the server is busy, the arrival is just added to the queue; if the server is idle, the arriving customer enters service and a departure time is determined based on a random sample from the service time distribution. The time of the departure event is placed in the events list.

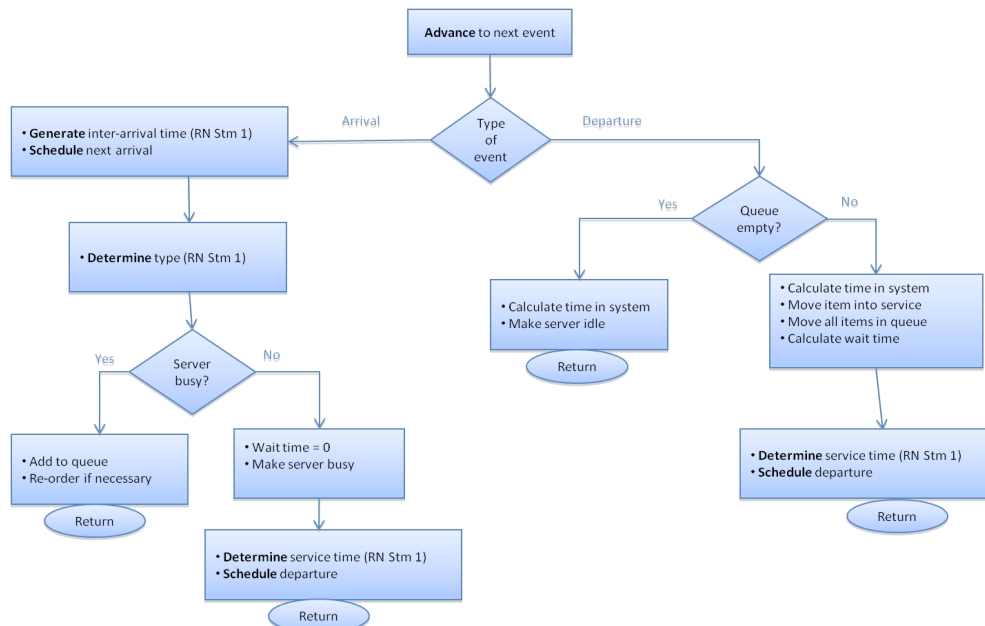


Figure 9.13 Simplified discrete-event logic

Similarly, if the current event is a departure (end of service), then the actions depend on whether or not the queue is empty. If the queue is empty, the server is made idle, if there are customers in the queue, the first customer is moved into service and a departure time is determined based on a random sample from the service time distribution. The departure event is placed in the events list.

After the processing of each event is completed the events list is sorted in ascending order in terms of time and the next event in the list is processed. This procedure is illustrated through the following manual simulation example of a simple single-server model of Joe's Copy Shop.



Figure 9.14 Joe's Copy Shop

The time between arrivals (inter-arrival times, IAT) of copy orders are exponentially distributed with a mean of 10 minutes. There are two types of copy jobs: commercial and personal. Joe puts commercial orders in red containers and personal orders in blue containers; thus the two order types (denoted by the variable *Type*) are referred to as Red and Blue.

On the average, 40% of the orders are commercial (Red) and the remaining 60% are personal jobs (Blue). On the average, the service time (ST) for Red jobs are twice as long as Blue (12 minutes versus 6 minutes). The time to copy is assumed to be exponentially distributed. Joe processes the orders in the sequence in which they arrive. Based on the discussions in the previous sections, the equations that are used to sample from probability distributions are as follows:

$$IAT_i \sim \exp(10) \rightarrow IAT_i = -10 \ln(1-u_i)$$

$$Type_i \sim \text{discrete empirical (40\% Red, 60\% Blue)}$$

Type, x	p(x)	P(x)	RN
Red	0.40	0.400.0	0 – 0.39
Blue	0.60	1.00	0.40 – 0.99

$$ST_i \sim \exp(12) \text{ if } Type = \text{Red} \rightarrow ST_i = -12 \ln(1-u_i)$$

$$ST_i \sim \exp(6) \text{ if } Type = \text{Blue} \rightarrow ST_i = -6 \ln(1-u_i)$$

A table is used to carry out the manual simulation. Figure 9.15 shows the first event to be considered in the manual simulation, which starts at time 100; that is, assume the simulation has been running for 100 minutes of simulated time when this example starts. Just prior to time 100, the server is idle and there are no items (jobs) in the system. This condition is referred to as “empty and idle”; in other words, the queue is empty and the server is idle.

An event occurs at time 100: the arrival of the 10th entity to the system. In this case, it is the 10th copy order. As in the flowchart in Figure 9.13, the arrival event results in three actions: (1) determining the arrival time of the next arrival, (2) determining the type of the current order, and, since the server (copier) is idle, (3) determining the service (copy) time of the 10th order. The last action will not happen if an arrival occurs when the server is busy; this is because service times are sampled and determined when an item enters the service process. In this case, the arrival passes through the queue, spends no time waiting, and goes directly into service—the copy job is started as soon as it arrives.

All three of the above actions require sampling from probability distributions. This is accomplished using the equations provided earlier and U(0,1) random numbers from a linear congruential generator. The parameters of the LCG are not provided here, just the results of executing the generator; in Figure 9.15 the values are shown in the table column labeled RN Stream 1. The random numbers are used to provide values for IAT, Type, and ST. The value of Type determines which ST equation to use. The sampled IAT value is added to the current simulation clock time (100) to generate a future event (arrival of the 11th job). ST is also added to the current clock time to generate another future event (the end of service for the 10th job, or the time it is completed). These two future events are shown in the Time column of the table.

When an item enters service, its wait time in the queue is calculated based on the current simulation time and the time when the item arrived; in this case, they are the same, and the item's wait time is 0. The item's time in the system is also calculated in a similar manner: departure time minus arrival time. In this case, job 10 is in the system 9.5 minutes. Normally the time in the system statistic is not calculated until the end of service event occurs in case the service is delayed due to an intervening downtime; however, in this simple model, it is assumed the server is always available. This information is summarized in Figure 9.15.

Time	Event	Action	RN Stm 1	Value	Time	Queue	FIFO					
100.0	Arr 10	IAT 11	0.0801	0.8	100.8					Service	Wait	Time
		Type 10	0.27708	Red	N/A					Time	Time	In Sys
		ST 10	0.54804	9.5	109.5							
							10 Red	N/A		9.5	0	9.5

Figure 9.15 Manual simulation—1st event

Once the future events are scheduled and statistics are calculated, the simulation jumps ahead in time to process the next event, as shown in Figure 9.16. The next event occurs at time 100.8 and is the arrival of the 11th job, which results in two

Carrying out this simple simulation manually provides insight into how randomness is simulated.

actions, again according to the flowchart in Figure 9.13: (1) determining the arrival time of the next arrival, and (2) determining the type of the current order. Since the server (copier) is busy, a third action, determining the service (copy) time the 11th order, is not performed as it was above. The service time will be determined when the 11th job can begin service. The Time column in the figure becomes an active events list; thus, 100.8 is dropped and 100.9, the arrival time of the 11th arrival, is added. The Queue column indicates which jobs are waiting to be processed and their position in the queue.

Time	Event	Action	RN Stm 1	Value	Time	Queue
100.0	Arr 10	IAT 11	0.0801	0.8	100.8	
		Type 10	0.27708	Red	N/A	
		ST 10	0.54804	9.5	109.5	
100.8	Arr 11	IAT 12	0.01418	0.1	100.9	
		Type 11	0.35148	Red	N/A	11 1st

Figure 9.16 Manual simulation—2nd event

As shown in Figure 9.17, the next event is the arrival of the 12th job. The actions generated by this event are the same as those of the previous event. Since a FIFO queue discipline is used, the 12th job is added to the end of the queue.

Time	Event	Action	RN Stm 1	Value	Time	Queue
100.0	Arr 10	IAT 11	0.0801	0.8	100.8	
		Type 10	0.27708	Red	N/A	
		ST 10	0.54804	9.5	109.5	
100.8	Arr 11	IAT 12	0.01418	0.1	100.9	
		Type 11	0.35148	Red	N/A	11 1st
100.9	Arr 12	IAT 13	0.70142	12.1	113.0	
		Type 12	0.46083	Blue	N/A	12 2nd 11 1st

Figure 9.17 Manual simulation – 3rd event

The fourth event, as shown in Figure 9.18, is the completion of 10th job (departure of the 10th item from the system). This causes the 11th job, the first one in the queue, to move into service. The service or copy time ST is determined based on the ST equation for the Red jobs and the random number in the RN Stream 1 column that is provided by generator. ST is added to the current clock time to generate the future event corresponding to the end of service for the 11th job; the time for this event to occur (115.3) is noted in the Time column.

As described earlier, the wait time and time in system for items are calculated when they enter service. In this case, the 11th job waited 8.7 minutes for service and was in the system 14.5 minutes when it left at time 115.3. This information is summarized in on the right side of Figure 9.18.

Time	Event	Action	RN Stm 1	Value	Time	Queue
100.0	Arr 10	IAT 11	0.080104	0.8	100.8	
		Type 10	0.277081	Red	N/A	
		ST 10	0.548039	9.5	109.5	
100.8	Arr 11	IAT 12	0.014175	0.1	100.9	
		Type 11	0.351481	Red	N/A	11 1st
100.9	Arr 12	IAT 13	0.701418	12.1	113.0	
		Type 12	0.460833	Blue	N/A	12 2nd 11 1st
109.5	Dep 10	ST 11	0.384065	5.8	115.3	12 1st

FIFO					
Type	IAT	Service Time	Wait Time	Time In Sys	
10 Red	N/A	9.5	0	9.5	
11 Red	0.8	5.8	8.7	14.5	

Figure 9.18 Manual simulation – 4th event

The process continues with the simulator processing the next event in the list. As shown in Figure 9.19, the next two events (5 and 6) are arrivals, and the jobs are just added to the queue. This is followed by two departure events. This example illustrates the logic involved with two basic types of events—arrival and departure—common in discrete-event simulation of queuing systems.

Another basic event is the end-of-simulation event. It is placed on the events list at the beginning of the simulation, and the simulation stops when that event is processed. In the table in Figure 9.19, if an end-of-simulation event had been specified as 115, then the last two events in the table would not have been executed, even though they were scheduled. You should review the table carefully to ensure you understand the logic of how a simulator works, how it generates random samples from the specified probability distributions, and how basic performance measures are calculated.

Time	Event	Action	RN Stm 1	Value	Time	Queue
100.0	Arr 10	IAT 11	0.080104	0.8	100.8	
		Type 10	0.277081	Red	N/A	
		ST 10	0.548039	9.5	109.5	
100.8	Arr 11	IAT 12	0.014175	0.1	100.9	
		Type 11	0.351481	Red	N/A	11 1st
100.9	Arr 12	IAT 13	0.701418	12.1	113.0	
		Type 12	0.460833	Blue	N/A	12 2nd 11 1st
109.5	Dep 10	ST 11	0.384065	5.8	115.3	12 1st
113.0	Arr 13	IAT 14	0.107706	1.1	114.1	
		Type 13	0.551673	Blue	N/A	13 2nd 12 1st
114.1	Arr 14	IAT 15	0.357665	4.4	118.5	
		Type 14	0.451961	Blue	N/A	14 3rd 13 2nd 12 1st
115.3	Dep 11	ST 12	0.340712	2.5	117.8	14 2nd 13 1st
117.8	Dep 12	ST 13	0.726916	7.8	125.6	14 1st

FIFO					
Type	IAT	Service Time	Wait Time	Time In Sys	
10 Red	N/A	9.5	0	9.5	
11 Red	0.8	5.8	8.7	14.5	
12 Blue	0.1	2.5	14.4	16.9	
13 Blue	12.1	7.8	4.8	12.6	

Figure 9.19 Manual simulation through the 8th event

In summary, the sequence of events that is processed by a simulator is determined by the values sampled from the probability distributions and the logic built into the model. Thus, both the probability distributions and logic drive the output results

from the simulation model. For example, the results would be different if different distributions were used or if the distributions used different parameter values. Similarly, the results would be different if the queue discipline was such that Blue items were always processed first, instead of using the order of arrivals. The example also illustrates the way time is handled in a discrete-event simulator—during the simulation, clock time does not progress in “ticks” like a regular clock, but it jumps to the time that the next event on the list occurs.

Figure 9.20 shows the events list in *FlexSim* for the Joe’s Copy Shop example. In the top portion of the figure there are two events awaiting processing at simulation time 71.880. The first is a departure or end of service event that is to occur at time 73.321; and, the second is an arrival event – the next customer is scheduled to arrive at time 85.496. The lower portion of the figure shows all of the actions that occurred at the current event (time 71.880) when a customer arrived. There are more actions listed in the figure than in the manual example since *FlexSim* can consider many more types of actions, such as OnEntry, OnExit, SendToPort triggers.

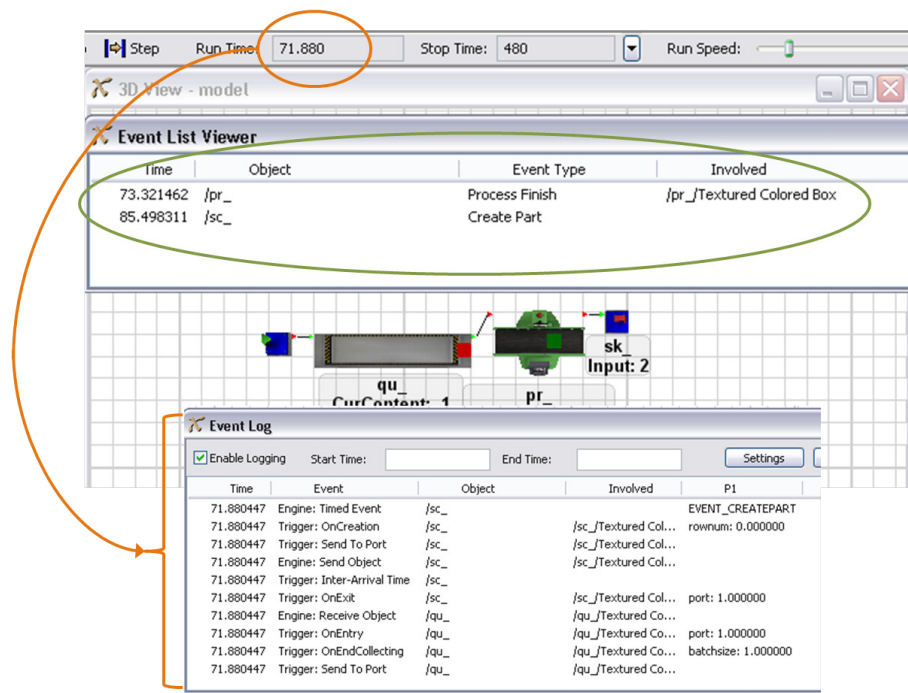


Figure 9.20 Events management in *FlexSim*

Section 9-9 Reducing variability in the samples

As noted above, the simulation results would be different if, for example, the queue discipline was changed. For example, based on production control theory, we would expect system throughput to increase if the shortest-processing-time (SPT) rule is used for the queue discipline instead of FIFO.

To illustrate how the simulation would differ if SPT is used instead of FIFO, consider the example from the brief manual simulation discussed in the preceding section. A portion of the simulation using FIFO is detailed in Figure 9.19. In Figure 9.21, a similar simulation is carried out using the same probability distributions and

the random number generator, except SPT is used as the queue discipline. The first difference to occur, which is on the third event, is indicated in red and boxed in the figure. In this case, the 12th job goes to the front of the queue upon its arrival instead of the back of the queue because it is a Blue item and has a shorter expected processing time.

Time	Event	Action	RN Stm 1	Value	Time	Queue
100.0	Arr 10	IAT 11	0.080104	0.8	100.8	
		Type 10	0.277081	Red	N/A	11 1st
		ST 10	0.548039	9.5	109.5	
100.8	Arr 11	IAT 12	0.014175	0.1	100.9	
		Type 11	0.351481	Red	N/A	
100.9	Arr 12	IAT 13	0.701418	12.1	113.0	
		Type 12	0.460833	Blue	N/A	12 1st 11 2nd
109.5	Dep 10	ST 12	0.384065	2.9	112.4	11 1st
112.4	Dep 12	ST 11	0.107706	1.4	113.8	queue empty
113.0	Arr 13	IAT 14	0.551673	8.0	121.0	
		Type 13	0.357665	Red	N/A	13 1st
113.8	Dep 11	ST 13	0.451961	7.2	121.0	queue empty
121.0	Arr 14	IAT 15	0.340712	4.2	125.2	
		Type 14	0.726916	Blue	N/A	14 1st
121.0	Dep 13	ST 14	0.334015	2.4	123.4	queue empty

Figure 9.21 Manual simulation using SPT, not FIFO

Since the simulation is using the same random number stream for all sources of variability, the 12th job has a different service time; in fact, all of the subsequent values have changed, including type of job, interarrival times, and service times. These changes cannot be attributed to a change in queue discipline. Consider, for instance, that the time it takes to copy the 12th order has nothing to do with the order in which it is processed. By using the same random number stream, additional variability has been introduced into the model and resulting analysis.

Variability can be inadvertently put into the simulation.

FIFO					SPT				
Type	IAT	Service Time	Wait Time	Time In Sys	Type	IAT	Service Time	Wait Time	Time In Sys
10 Red	N/A	9.5	0	9.5	10 Red	N/A	9.5	0	9.5
11 Red	0.8	5.8	8.7	14.5	11 Red	0.8	1.4	11.6	13.0
12 Blue	0.1	2.5	14.4	16.9	12 Blue	0.1	2.9	8.6	11.5
13 Blue	12.1	7.8	4.8	12.6	13 Red	12.1	7.2	0.8	8.0
14 Blue	1.1				14 Blue	8.0	2.4		
15	4.4				15	4.2			
Mean			7.0	13.4				5.3	10.5

Figure 9.22 Comparison of FIFO and SPT for short, manual simulation

A comparison on the results of the two short simulations is provided in Figure 9.22.

Random number
stream selection
is important to
understand.

Assigning a different random number stream to each source of variability can reduce this problem. If that were done in this short example, the item types would not change. In fact, in this short example, had a separate stream of random numbers been used for each source of variability, the output for the two simulations would have been exactly the same. Over a longer simulation, differences would have occurred; however, these differences would have been attributed only to the change in queue ordering.

In most simulation software, it is quite easy to use a different random number stream for each source of variability. Typically, as is the case in *FlexSim*, one additional parameter is required when specifying the probability distribution. For example, in *FlexSim* the command `normal(10, 2, 8)` instructs the software to sample from a normal distribution with a mean of 10 and a standard deviation of 2 using random number stream 8.

This approach is referred to as common random numbers and is an example of a variance reduction technique. Since these techniques reduce the variability in comparisons among the models, they also reduce the number of replications required to attain a certain level of precision, or they result in a more precise estimate (narrower confidence interval) for the same number of replications. An example of the effect this has on a comparison and further discussion is provided in Chapter 10.

References

1. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.

Chapter 9 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. All of the distributions below have the same mean. Rank them in order of increasing variability.
 - a. Exponential (100)
 - b. Normal (100, 20)
 - c. Triangular (80, 130, 90)
 - d. Uniform (50, 150)
3. Identify an operations system and select two random variables that would be a part of a simulation model. Describe how you would decide what distribution to use to represent each of the random variables.
4. The time between failures on a machine is exponentially distributed with a mean of 1000 hours. Generate a random time to failure using a $U(0,1)$ value of 0.1209.
5. The time to repair a failed machine is uniformly distributed between 5 and 15 minutes. Generate a random repair time using a $U(0,1)$ value of 0.7344.
6. The time to travel between two locations is normally distributed with a mean of 5 minutes and a standard deviation of 2 minutes. Generate a random travel time using a $U(0,1)$ value of 0.9056.
7. The density function for the time to perform a blood test, in minutes, is. Generate a random test time using
8. Service times were found to have the density $f(x) = \frac{x}{12}$ for $1 \leq x \leq 5$. Generate a random service time using $u_1 = 0.3132$
9. The distribution of the number of tests to be performed on a patient's blood sample is provided below. How many total tests are performed on the first four patients? Use $u_1 = 0.2483$, $u_2 = 0.3637$, $u_3 = 0.0997$ and $u_4 = 0.6424$.

1	0.10
2	0.35
3	0.35
4	0.20
10. The probability that an undergraduate student with a certain major will request assistance at a help desk is provided below. Using $u_1 = 0.5132$ and $u_2 = 0.7112$, generate two types of student majors.

Arts & Sciences	0.35
Business	0.25
Engineering	0.25
Other	0.15

11. For the LCG with parameters $a=13$, $c=5$, $m=32$, and $N_0=4$, generate two $U(0,1)$ random numbers.
12. For the generator described in question 11, what is the maximum number of values that will be generated before the pattern repeats?
13. For an online bookseller, the time between customer orders is exponentially distributed with a mean of 15 minutes. Using $N_i = (27N_{i-1} + 4) \bmod(64)$ and $N_0 = 10$, generate the time the next two orders arrive, assuming the last arrival occurred at time 100.
14. The time to assemble an item is uniformly distributed between 10 and 20 seconds. Generate the next two assembly times using the generator:

$$N_{i+1} = (10N_i + 2) \bmod(32), \text{ with } N_0 = 16$$
15. In the template provided below, continue the manual simulation of Joe's Copy Shop for the next three events.

										Arrive	Service	Dep	Wait	Time
Time	Event	Action	RN Stm 1	Value	Time	Queue		Type	IAT	Time	Time	Time	Time	In Sys
117.8	Dep 12	ST 13	0.726916	7.8	125.6	14 1st	10	Red	N/A	100.0	9.5	109.5	0	9.5
			0.334015				11	Red	0.8	100.8	5.8	115.3	8.7	12.6
			0.959562				12	Blue	0.1	100.9	2.5	117.8	14.4	16.9
			0.608115				13	Blue	12.1	113.0	7.8	125.6	4.8	12.6
			0.450250				14	Blue	1.1	114.1				
			0.318900				15		4.4	118.5				
							16							
							17							
							18							

16. Describe how you would implement common random numbers in a model that uses the following distributions:
 - Time between arrivals: exponential(0, 20)
 - Process time for Step A: normal(12,3)
 - Process time for Step B: normal(5,1)
 - Probability of rework: bernoulli(10, 1, 0)
 - Time between failures: exponential (0, 500)
 - Time to repair: triangular(10, 20, 12)

Chapter

10

Analyzing Simulation Output

A simulation model is developed in order to generate output for analysis. Since models are representations of real systems, they are used to understand the behavior of those systems. Some of the inputs to a model are values for decision variables (i.e., values that are controlled by the decision maker). Output from a model provides information on the consequences of setting the decision variables in a certain manner. A model is like a laboratory, and as such, experiments are designed and run using models.

This chapter begins with a discussion of the definition of key tactical experimental design parameters. It is followed by sections covering

- two types of systems, terminating and non-terminating, and their impact on setting the design parameters;
- a means for determining how many replications of a model are needed;
- *FlexSim's* Experimenter, a very powerful means for conducting experiments with simulation models;
- how to compare output obtained from two simulated alternatives;
- the variance reduction technique common random numbers (CRNs);
- how to perform multiple comparisons between more than two alternatives.

Careful statistical analysis is paramount when using output from a discrete-event simulation model. In order to draw conclusions and make valid inferences, simulation models must be replicated; that is, the models must be run multiple times. This involves independent random samples provided from the various statistical distri-

butions that can be used in a model. The information gathered from the multiple runs is combined to provide a basis for conclusions and inference. For example, confidence intervals are used to estimate the means of output or response variables which provide the foundation for inference and decision making. Methodologies for making inferences from simulation models are discussed later in the chapter, following a discussion of some important tactical experimentation variables that must be set in order for the model to generate the data needed for analysis.

As illustrated in Figure 10.1, it is important to distinguish between replications, scenarios, and experiments. A replication is a single run of a simulation model for a specified time. A scenario is a set of variables for which values are changed and performance is recorded: each scenario is replicated multiple times. An experiment is a set of scenarios.

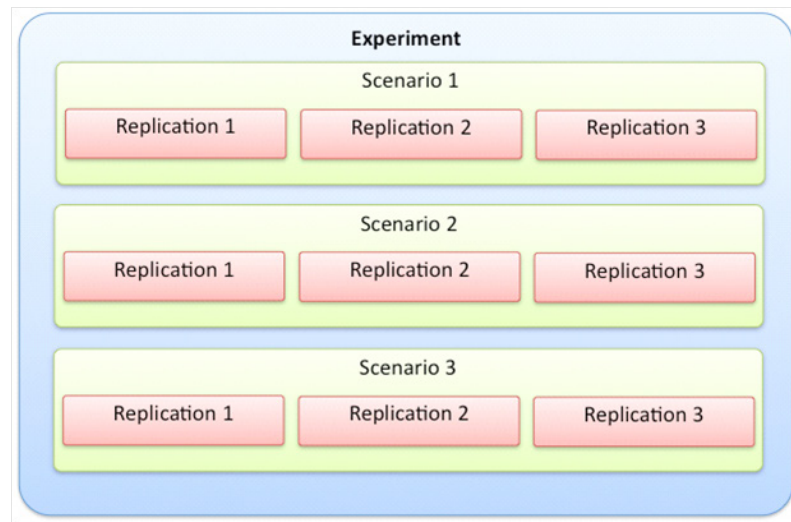


Figure 10.1 Replications, scenarios, and experiments

Section 10-1 Key tactical experimental design parameters

Before a model can be run or replicated, a few experimentation parameters need to be set: the number of replications, starting or initial conditions, and run length (how long the simulation is run in each replication).

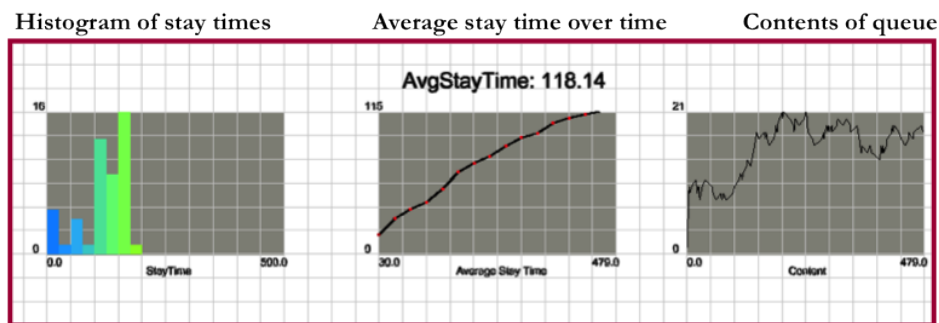
The importance of these parameters is illustrated here through a sensitivity analysis on a simple system. The analysis considers average stay time (wait time) in the queue as the performance measure, or output variable of interest. In addition to only reporting the mean stay time for each condition at the end of the simulation, a histogram of the distribution of stay times over the duration of the simulation, a plot of the average stay time over time, and a plot of the number of items in the queue over time are provided in order to give more insight into the system's behavior.

There are a number of parameters that have to be set before experimentation can begin.

The sensitivity analysis is based on a simple, single-server, FIFO queuing system with infinite queue capacity, and the server is always available (no downtime). The inter-arrival times are exponentially distributed with a mean of 10 minutes, and service times are exponentially distributed with a mean of 9 minutes.

Figure 10.2 illustrates the importance of replications. In both panels, the system begins with 10 items in the system at time 0, and the model is run for 8 hours. The top panel shows the results from the first replication and the lower panel shows the results from the fourth replication. These are just two possible ways, out of an infinite number of possible ways, the system could behave in an 8-hour period. Obviously, there is a large difference in the performance measure; the average stay time for the first replication (118.1 minutes) is nearly twice that of the fourth replication (66.3 minutes). This difference indicates that there is considerable variability in the system. Thus, inferring the behavior of the system based on one of these replications would likely result in a different conclusion and a different decision on what the values of the decision variables should be.

Replication number: 1
Initial number in system: 10
Simulation run length: 8 hours



Replication number: 4
Initial number in system: 10
Simulation run length: 8 hours

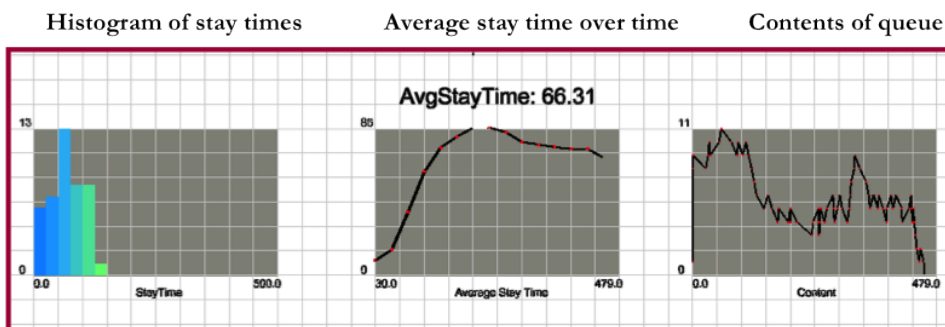


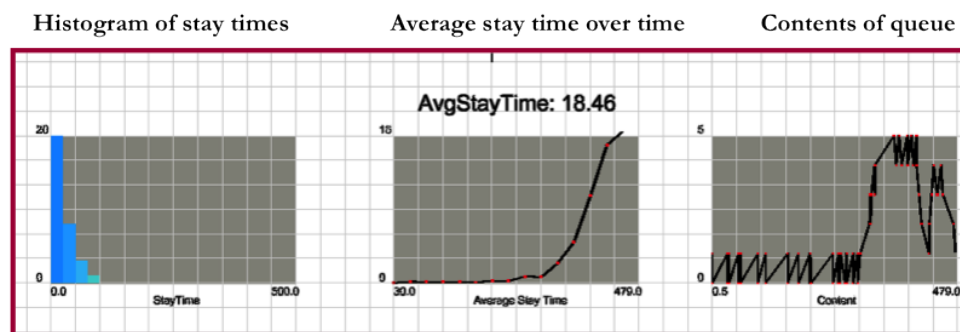
Figure 10.2 Importance of replications

Replications are necessary for an appropriate analysis of a simulation.

In order to avoid making inferences on a single, possibly extreme value, the model is replicated a number of times, and the values of the performance measures are combined, typically averaged. Determining the number of replications to run is the subject of a later section in this chapter.

Figure 10.3 illustrates the importance of starting, or initial, conditions. In both panels, replication 4 is used and the model is run for eight hours. The top panel shows the results if the simulation begins with no items in the system. The bottom panel (same as the bottom panel in the previous figure) shows the results if the system begins with ten items in the system. Obviously, there is a large difference in the performance measures.

Replication number: 4
Initial number in system: 0
Simulation run length: 8 hours



Replication number: 4
Initial number in system: 10
Simulation run length: 8 hours

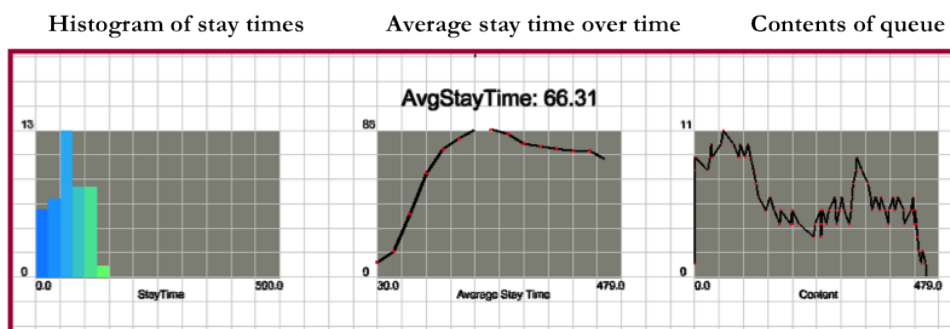


Figure 10.3 Importance of starting or initial conditions

Initial conditions have to be set for each replication.

The average stay time when there are no items in the system at time 0 (18.5 minutes) is less than a third of the stay time when there are 10 items at time 0 (66.3 minutes). This indicates that the model is very sensitive to the starting conditions. In general, the shorter the simulation run, the more the starting condition

impacts performance measures. Thus, inferring the behavior of a system based on the different starting conditions would likely result in a different conclusion and a different decision on what the values of the decision variables should be.

Figure 10.4 illustrates the importance of the run length or duration of the simulation. In this figure there are four panels all based on replication 4. The top two panels start the simulation with no items in the system and the bottom two panels start the simulation with 10 items in the system. The left two panels are run for 8 hours and the right two panels are run for 800 hours.

The length of a simulation run can impact the results.

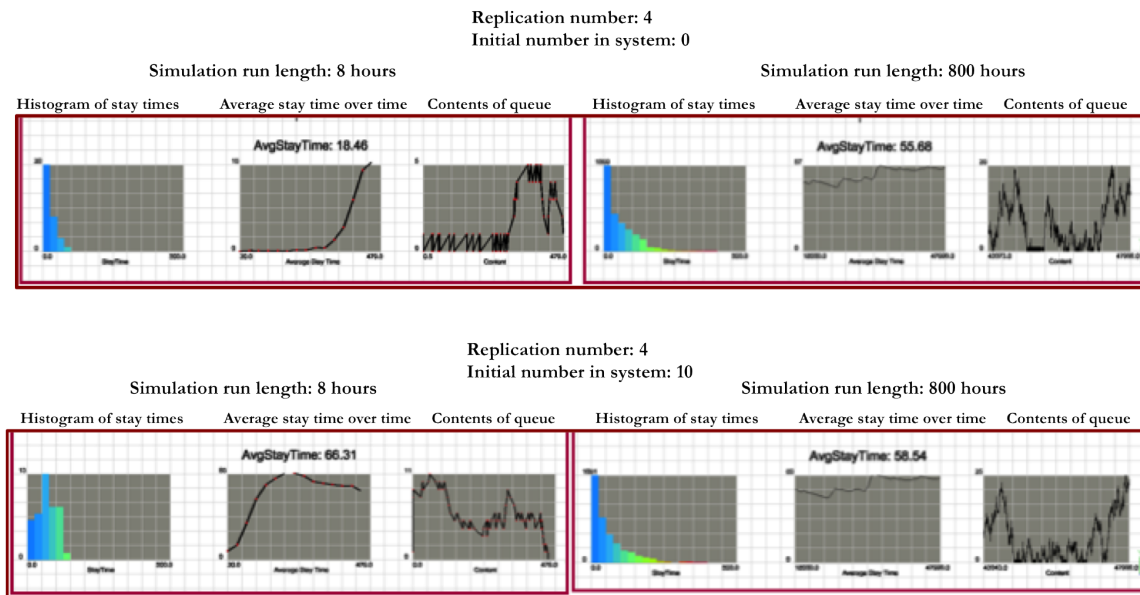


Figure 10.4 Importance of duration of simulation or run length

The average stay times for these four conditions are summarized in Table 10.1.

		Run length	
		8 hours	800 hours
Initial number in system	0	18.5 min.	55.7 min.
	10	66.3 min.	56.5 min.

Table 10.1 Average stay times

Note that in the figure and in the table, when the two options (initial number in the system 0 and 10) are run for 800 hours, there is little difference in the average stay time. This is to be expected; the longer a simulation model is run, the less starting conditions affect the performance measures. This is because the longer a model is run, the closer it gets to steady state conditions (if steady-state conditions exist). Steady state does not mean the individual stay times become constant; rather that the *distribution* of the stay times becomes constant and thus the distribution's mean approaches a constant.

Also note that the duration has a significant affect on the average stay time when the initial number in the system is 0, but has much less affect when there are 10 items in the system at time 0. In this particular system, the steady-state average number of items in the system is near 10. Therefore, when the initial condition is 10, the model starts near the system's steady-state values. As discussed above, typically, if the model is run long enough, regardless of starting conditions, the performance measures converge to their steady state values; therefore, before experiments can be run and the output analyzed, the number of replications to run, the initial or starting conditions, and the run length must all be determined. Subsequent sections, after an introduction to terminating and non-terminating systems, provide guidance on these issues.

The type of analysis varies between terminating and non-terminating systems.

Section 10-2 Terminating versus non-terminating systems

The importance of experimentation parameters and the means to set them as described above depend on the type of system being modeled—terminating or non-terminating.

Analysis of terminating systems

As shown in Figure 10.5, terminating systems have well defined starting and ending points that cover a period of interest during which performance is measured. Typically, these points correspond to specific times. For example, a clinic is interested in patient wait time during its 8 hours of operation, or a restaurant is interested in the time customers wait for a table during a 2.5 hour lunch rush period.

For terminating systems, the focus is on performance over a particular period of time. Typically, the system does not reach steady-state conditions during this period. For example, patients' wait times in a clinic most likely do not reach steady state during the clinic's operating hours, because the clinic is not open long enough for most performance measures to reach steady state; therefore, it would be inappropriate to use steady-state measures, such as those from queuing theory or running a simulation model for a very long period, to assess performance and compare alternatives.

Setting the experimentation parameters for terminating systems is typically quite straightforward. The run length is usually well defined

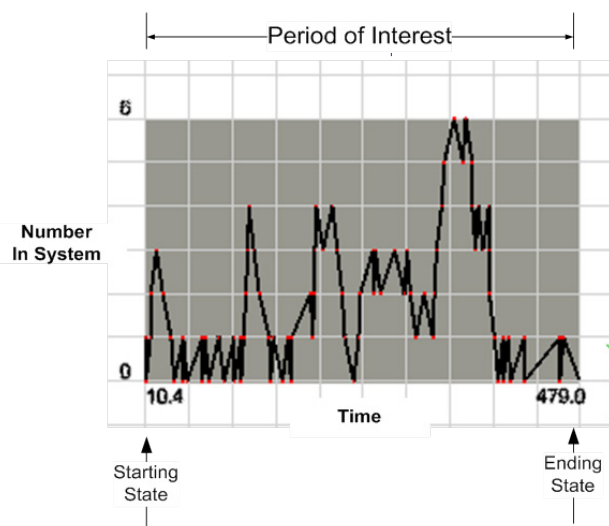


Figure 10.5 A state variable in a terminating system

by the system being modeled; that is, the simulation is run until the termination event occurs, such as the closing of a clinic or the end of a restaurant's rush period. However, the closing of the clinic may not be the terminating event; the closing may only be when the arrivals stop. The simulation may continue beyond the closing time until all patients are served. In most simulation software, when there are no events in the events list to process—the arrival process is stopped and all items have exited the model—the simulation stops running.

The initial or starting conditions are also clear from the system's data. Oftentimes they are referred to as *empty and idle*—at time 0, the system contains no flowitems and all of the resources are idle. As in the clinic example, this is typically the ending or terminating state. The simulation ends when all patients are processed and as a result all of the resources become idle. Note that it is plausible that there could be a pulse of flowitems in the system at the start of the simulation (i.e., patients waiting for the clinic to open). In that case, the model would generate the typical number of items at time 0.

The number of replications needed depends on how precise the analysis needs to be and how much variability there is in the system. The method for determining the number of replications is explained in a later section.

Analysis of non-terminating systems

For non-terminating systems, simulation models are typically run for a very long duration because, as the name implies, there is no known ending or terminating point. In non-terminating systems the focus is typically on long-run or steady-state performance.

As shown in Figure 10.6, non-terminating systems go through a transient period and then possibly reach steady state. As stated earlier, steady state does not mean that individual values, such as wait time, converge to a constant; it means that the *distribution* of the performance measure settles down and approaches a constant. This is illustrated at the bottom of Figure 10.6 where the distribution of the state variable is quite skewed early in the simulation, but becomes symmetrical as the simulation progresses through the transient period into steady state.

Setting the experimentation parameters for non-terminating systems is nowhere near as straightforward as it is for terminating systems. The run length of the simulation is most problematic to determine as there isn't one single agreed-upon approach for successfully determining it in non-terminating systems and many of the techniques require sophisticated analysis techniques. A later version of this text may explore some of those approaches; however, for now, we suggest using a series of plots, one for each performance measure of interest, similar to Figure 10.6. We also suggest the plots include at least a few replications. For more information on output analysis of non-terminating systems see, Law¹ or Robinson.² As shown in Figure 10.6, the plot should indicate a time when the performance measure tends to level off. This point in time is designated as the *warmup time*. When the simulation reaches

the warmup time, all statistical counters are reset; the reported output statistics are based on the time period beginning with the warmup time and ending with the completion of the simulation run. Note that the conditions in the model are not reset, just the statistical counters. Removal of data from the warm-up time reduces the startup bias.

Starting conditions oftentimes do not have a significant effect on performance measures in non-terminating systems because of their long run times—this is especially true if the warm-up approach is used. The method for determining the number of replications to run is discussed in the next section and can be applied to non-terminating systems; however, since models of non-terminating systems have long run times, typically few replications are run.

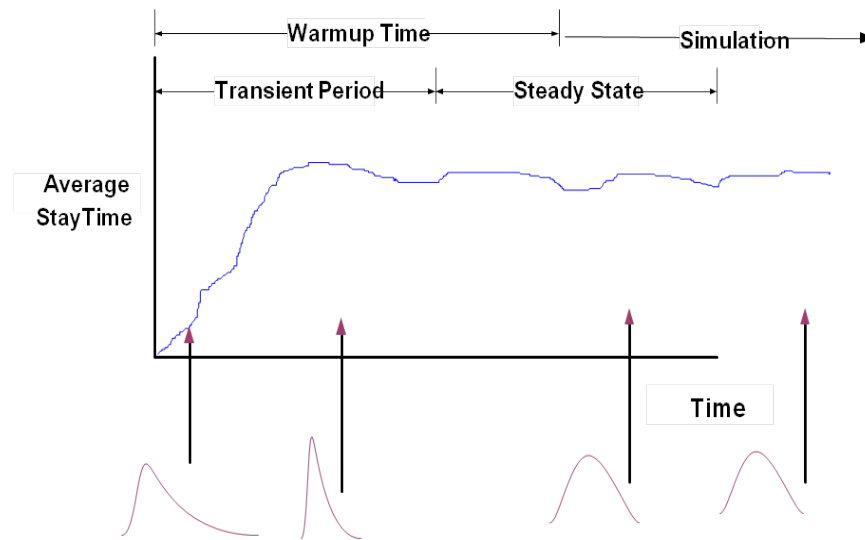


Figure 10.6 A state variable in a non-terminating system

One issue that often arises is the relationship between simulated time and real time. As shown in the timeline (labeled System A) at the top of Figure 10.7, if the operation being simulated runs 24/7, then system time and simulated time are the same. System A might represent a continuous manufacturing operation in which workers come and go during the day but operations continue. The start of one 24-hour period picks up where the end of the last stopped.

In Figure 10.7, System B might represent a small woodworking shop which operates 8 hours a day. At the end of the day, work stops but picks up again the next day where it left off. Since there is typically no need to simulate the time the shop is closed, the simulation may run the work time continuously and skip the closed time. In this case 8 hours of simulation time might actually represent 24 hours of system time. Therefore, in this case, to run a week of operation only 40 hours of simulated time is needed and not 168 hours of clock time.

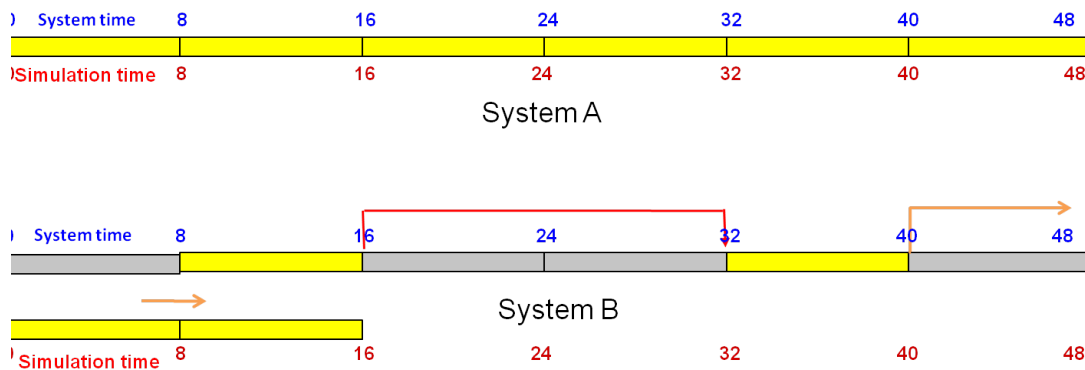


Figure 10.7 Simulation time for non-terminating systems

Section 10-3 Determining the number of replications

As discussed earlier, in order to avoid making inferences on a single, possibly extreme value, a model should be replicated a number of times and the values of the performance measures from each replication averaged. The general intent is to conduct experiments with the simulation model and estimate the population performance measures of interest, such as the mean. As shown in the formula below, the expected value of the population mean, μ , is estimated by the mean of the sample means that are obtained from each replication. Similarly, a measure of the dispersion in the estimates is given by the variance in the sample means.

$$E(\mu) = \bar{X} = \frac{1}{n} \sum_{i=1}^n \bar{X}_i$$

$$E(\sigma^2) = S^2 = \frac{1}{n-1} \sum_{i=1}^n (\bar{X}_i - \bar{X})^2$$

Since the value of the population mean will never be known with certainty, the more runs that are used to estimate the mean the better; that is, the larger the sample size, the better we feel about the estimate. There is, however, a price to obtain an estimate that is based on a large number of replications. A complex model may take a long time to run, thus there may not be time to run a model many times, especially if there are many options or scenarios to consider.

Typically, the individual values that comprise the mean for each replication, such as an individual customer's wait time, are correlated. For example, if one customer's wait time is long, then it is likely that those before and after that customer had long wait times as well; however, the mean values for each run or replication are independent. This is because in discrete-event simulation software, each source of randomness uses independent random number values to do the sampling.

The appropriate number of replications may not be obvious.

Automating the experimentation process can speed simulation analysis.

The mean is a point estimate in that it does not provide any information on the precision of the estimate being made. From basic statistics, a confidence interval, as defined below, is used to indicate the level of precision of an estimate:

$$\bar{X} - t_{\alpha/2, n-1} \frac{S}{\sqrt{n}} \leq \mu \leq \bar{X} + t_{\alpha/2, n-1} \frac{S}{\sqrt{n}}.$$

That is, using the above equation provides an interval estimate of the population mean μ such that we are $(1-\alpha)\%$ confident that the true mean is between the calculated limits. If the width of the interval is too wide, then the number of replications is typically increased.

In order to determine the number of replications needed, the desired precision (as indicated by the half-width of the confidence interval) is specified. Using the confidence interval expression above, the half width, e is

$$e = t_{\alpha/2, n-1} \frac{S}{\sqrt{n}}.$$

As shown in the following equation, the number of replications required is obtained by solving the above half-width equation for n . To simplify the solution (not having n on both sides of the equation), the standard normal distribution is substituted for the t-distribution. Recall from statistics that the two distributions become very similar if the sample size, n , is large (typically about 30). An alternative form of the number of replications formula is provided below. The alternative is used when it is easier to specify the desired precision relative to the mean rather than as an absolute value. That is, instead of specifying the precision of the estimate in terms of e (e.g. within 5 minutes of the true mean, $e = 5$), precision is specified via r , e.g., within 5%, $r = 0.05$, of the true mean.

$$n = \left(\frac{Z_{\alpha/2} S}{e} \right)^2 = \left(\frac{Z_{\alpha/2} S}{\frac{r\bar{X}}{1+r}} \right)^2$$

Section 10-4 Creating experiments using FlexSim's Experimenter

FlexSim provides a very powerful analysis tool: the Experimenter. The Experimenter is found in the Statistics drop down menu in the main toolbar and key interfaces are shown in Figure 10.8. The Experimenter is configured by stepping through the tabs on the main interface:

- *Scenarios*: Alternative model configurations or conditions that are being considered
- *Performance measures*: Key variables that are used to decide which scenario is preferred

- *Experiment Run*: Run time, number of replications, and length of the warm-up period, if any
- *Advanced*: Custom logic for additional control of scenarios and replications

Once all parameters have been defined, the Reset Experiment button is pushed and the Experimenter tracks the value of the specified performance measures for each replication within each scenario. When all of the simulation runs defined in the Experimenter are complete, a variety of outputs are available for analysis by pushing the View Results button at the bottom of the interface.

For example, in Figure 10.8 a single scenario is considered. The model was replicated 10 times with each replication being run for 7200 time units. Results for one performance measure, Awaiting Assembly, are displayed in the figure. Awaiting Assembly represents the time items wait prior to being processed at an assembly operation. The plot shows the mean value of the performance measure for each replication as well as basic descriptive statistics and a confidence interval estimate of the measure.

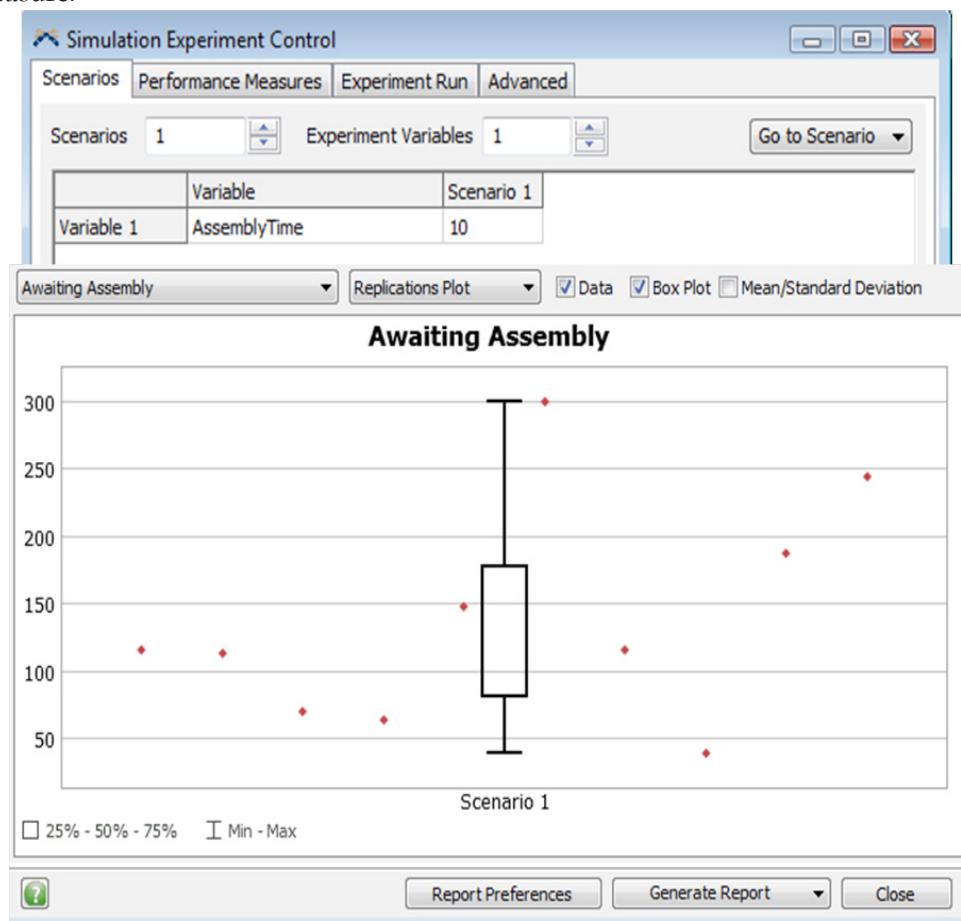


Figure 10.8 Interfaces to the Experimenter

Additional details for setting up the experimenter are included in the Appendix and *FlexSim*'s help system.

Section 10-5 Comparing two alternatives

A key use of simulation modeling and analysis is comparing alternatives in order to improve the design and operation of a system. A common approach to formally compare two alternatives is by statistically testing their means in order to see if there is significant difference. The following hypotheses are tested:

$$H_0 : \mu_1 = \mu_2$$

and

$$H_1 : \mu_1 \neq \mu_2.$$

If H_0 (the null hypothesis) is rejected, then it is concluded that there is a significant difference in the mean of the two populations that represent the two alternatives.

The table in Figure 10.9 summarizes the assumptions of three commonly used statistical comparison approaches. The two-sample t-test is commonly covered in introductory statistics courses. Welch's method is covered in many simulation books. The paired-t test is often covered in statistics book and simulation books, although not always.

	Two-sample t	Welch's	Paired t
Observations within population	Independently and normally distributed	Independently and normally distributed	Independently and normally distributed
Observations between populations	Independent	Independent	Do not need to be independent
Population variances	Equal	Do not need to be equal	Do not need to be equal
Number of observations in each population	Do not need to be equal	Do not need to be equal	Equal

Figure 10.9 Analysis of non-terminating systems

As shown in the table, the paired-t does not require the observations between populations to be independent. This is an important difference and a key reason why the paired-t is used quite often in simulation modeling and analysis. As was introduced in the last chapter and as will be discussed in more detail later in this chapter, the use of common random numbers in a simulation model oftentimes greatly reduces the variation in comparisons.

There are statistical approaches to compare alternatives.

When using common random numbers, the two-sample t and Welch's approach are not appropriate since they assume independence between populations or alternatives. One general drawback to the paired-t approach is that it requires the same sample size for all of the alternatives. When performing physical or human-based experiments, this is often a key restriction; however, in simulation modeling and analysis, it is typically not a problem. It is easy to make the number of replications the same for all alternatives.

In general, the paired-t test uses the following hypotheses:

$$H_0 : \mu_1 = \mu_2 \rightarrow \mu_1 - \mu_2 = 0 \rightarrow \mu_{12} = 0;$$

$$H_1 : \mu_1 \neq \mu_2 \rightarrow \mu_1 - \mu_2 \neq 0 \rightarrow \mu_{12} \neq 0.$$

The test can be performed by using confidence intervals. If the following interval includes the hypothesized value, 0 in this case, then the null hypothesis cannot be rejected and it is inferred, based on the sample results, that there is no significant difference between the two alternatives:

$$\left(\bar{d}_{12} - t_{\alpha/2, n-1} \frac{S_{12}}{\sqrt{n}} \leq \mu_{12} \leq \bar{d}_{12} + t_{\alpha/2, n-1} \frac{S_{12}}{\sqrt{n}} \right).$$

For each of the n replications, the difference in the performance measure of interest is calculated. The mean and standard deviation of those n differences are \bar{d}_{12} and S_{12} , respectively.

The following example illustrates the application of the approach. Consider the Joe's Copies example from the last chapter. Recall that Joe processes two types of copy jobs, commercial (Red) and personal (Blue). Blue jobs generally take less time than Red. Joe is trying to decide the best way to process the jobs, either in the sequence they arrive (FIFO) or Blue jobs first (shortest processing time, SPT).

Figure 10.10 shows the results from $n=20$ replications of a model using FIFO and a model using SPT. The performance measure of interest is throughput per day. In the first replication one more job is processed using FIFO than SPT; in the second replication two more jobs are processed using SPT, etc.

The mean and standard deviation of the differences, \bar{d}_{12} and S_{12} , are -2.4 and 6.7, respectively, which means that based on the simulation, we expect 2.4 more jobs to be processed per day using SPT. The difference in Figure 10.10 is calculated as FIFO - SPT and thus a negative value implies SPT > FIFO. Since the performance measure being considered is throughput, this is considered an improvement.

However, a 95% confidence interval estimate of the mean difference is

$$\left(-2.4 - 2.093 \frac{6.7}{\sqrt{20}} \leq \mu_{12} \leq -2.4 + 2.093 \frac{6.7}{\sqrt{20}} \right) \rightarrow -5.5 \leq \mu_{12} \leq 0.7$$

Using common random numbers (CRNs) can improve the precision of the estimates made by the simulation model.

This says we are 95% confident the true mean difference is between -5.5 and 0.7. Since this interval contains 0, the hypothesized value, we conclude there is no statistically significant difference in throughput by using FIFO or SPT.

	FIFO	SPT	Diff
1	48	47	1
2	38	40	-2
3	48	44	4
4	46	44	2
5	45	47	-2
6	46	49	-3
7	39	48	-9
8	45	44	1
9	37	42	-5
10	35	40	-5
11	30	38	-8
12	34	35	-1
13	36	43	-7
14	36	35	1
15	37	47	-10
16	37	45	-8
17	48	35	13
18	38	43	-5
19	32	47	-15
20	48	38	10
Mean			-2.4
Std Dev			6.7
n	20		
t	2.093024		
LL			-5.5
UL			0.7

Figure 10.10 Comparison in throughput, FIFO vs. SPT

Section 10-6 Variance reduction through common random numbers

The notion of variance reduction techniques, specifically common random numbers (CRNs), was introduced in the previous chapter. Using CRNs means that each source of variability in a model (e.g., time between arrivals, process time, quality indicator, time between failures) is assigned its own stream of random numbers. By using CRNs, the values between replications are no longer independent, and thus the paired-t test becomes the appropriate parametric statistical test.

	FIFO	SPT	Diff
1	44	48	-4
2	40	41	-1
3	46	49	-3
4	27	31	-4
5	43	43	0
6	41	47	-6
7	41	42	-1
8	47	50	-3
9	42	41	1
10	43	42	1
11	40	39	1
12	29	29	0
13	40	42	-2
14	40	48	-8
15	47	47	0
16	43	41	2
17	33	33	0
18	28	35	-7
19	36	35	1
20	36	37	-1
Mean			-1.7
Std dev			2.9
n			
LL			-3.0
UL			-0.4

Figure 10.11 Comparison in throughput, FIFO vs. SPT, using common random numbers

without using CRNs. This means CRNs provide a more precise estimate for the same number of replications, or that fewer replications can be used for the same level of precision. Also notice that the inference is different. Now, since the 95% confidence interval does not contain the hypothesized value 0, we conclude SPT results in significantly more daily average throughput.

Section 10-7 Multiple comparisons of alternatives

There are several approaches outlined in this section for comparing $k > 2$ alternatives. More information on these approaches can be found in *Simulation Modeling and Analysis* by Law¹, or earlier editions by Law & Kelton.

The k alternatives can be compared in a pair-wise manner to a standard or base case. In this situation, there are $k-1$ confidence intervals for $k-1$ differences.

A simple example can illustrate what happens when the values between replications are not independent. Consider that there is a large random number in a stream that results in a large time between arrivals for the 20th item in the first replication of the base case of a simulation. If this is the case, then by using CRNs, the 20th time between arrivals in the first replication for all of the alternatives will be large as well. Normally, although not always, this induced correlation will have a positive effect in that it will reduce the variability among the alternatives.

The effect of CRNs is illustrated using the Joe's Copies example from the previous section. A similar analysis is performed except that in this case, CRNs are used. The output from 25 replications of Joe's Copies model using FIFO and SPT is provided in Figure 10.11.

Notice that the standard deviation of the difference is considerably less than the previous results (2.9 versus 6.7). As a result, the 95% confidence interval, $-3.0 \leq \mu_{12} \leq -0.4$, is much narrower. The half-width of this interval is 1.3 compared with 2.1

At times more than two alternatives must be compared.

For example, $\mu_1 - \mu_2, \mu_1 - \mu_3$, etc., where μ_1 is the designated standard. If the overall desired level of confidence is $1-\alpha$, then each individual interval should be set so that its confidence level is

$$1 - \frac{\alpha}{(k-1)}$$

therefore, each interval's level of significance should be

$$\alpha/(k-1)$$

The probability of committing Type I Error is represented by α . In this case, a Type I Error means rejecting the hypothesis that there is no difference in the alternatives when in fact there is a difference in the populations.

The k alternatives can also be compared in pair-wise manner with every other alternative. In this case, there are $k(k-1)/2$ confidence intervals. If the overall desired level of confidence is $1-\alpha$, then each individual interval should be set so that its confidence level is

$$1 - \frac{\alpha}{k(k-1)/2}$$

therefore, each interval's level of significance should be

$$2\alpha/k(k-1).$$

Summary

This chapter provides an overview of performing statistical analysis on data obtained from discrete-event simulation models. It is critical that output from simulations are analyzed properly. No matter how “good” the model is, in other words, how well it represents the real system, improper analysis of the simulation output could, and will likely, lead to wrong recommendation and wrong decisions. As is pointed out in the chapter, much has been written in the simulation literature about analyzing simulation output. It is not our intent in this chapter to cover, or even survey, all of methodologies and techniques that are available and their nuances, but to provide a basic and applied foundation for analyzing simulation output.

The analysis approach that is used to analyze simulation output is heavily dependent upon the type of system being considered – terminating versus non-terminating. In either case, the model must be replicated or run more than once. Also, it is important to decide how long to run each replication and what the starting conditions are. Of course, as with any experimental design, the decision variables and performance measures must be identified and clearly defined before using simulation.

Since most analyses involve comparisons, the chapter provides basic methodologies for statistically comparing output (performance measures) from one or more

alternatives. In order to make comparisons more precise for a given number of replications, the chapter provides a simple approach for variance reduction, i.e., the use of common random numbers.

The chapter and its appendix discuss the use of *FlexSim's* Experimenter to facilitate analysis of simulation model output.

References

1. Law, A. (2007) *Simulation Modeling and Analysis*, New York: McGraw-Hill.
2. Robinson, S. (2004) *Simulation: The Practice of Model Development and Use*, Chichester, England: John Wiley & Sons, Ltd.

Chapter 10 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Briefly explain why simulations should be run more than one time; that is, conclusions should be based on more than one replication.
3. Discuss the difference between replications and scenarios.
4. Discuss the differences between terminating and non-terminating systems.
5. Which of the following would likely be modeled as terminating systems and which would be modeled as non-terminating systems:
 - a. customer wait time at a fast-food restaurant between 11 a.m. and 1 p.m.
 - b. inventory levels in a manufacturing cell that produces a commodity product;
 - c. throughput per week of a high-speed bottling line;
 - d. throughput per week in a job shop that produces custom products;
 - e. the time to produce a specified quantity of product;
 - f. patient wait time in a small, single doctor clinic;
 - g. the number of passengers waiting to clear security in an airport;
 - h. time between flight arrival and when baggage reaches its carousel;
 - i. the utilization of equipment used in aluminum production.

6. Describe how to implement the variance-reduction technique, Common Random Numbers.

7. The data to the right are average patient waiting times, in minutes. They were obtained from 15 replications of a simulation model of a hospital imaging department.

- Provide a 95% confidence interval estimate of the true patient wait time.
- Determine the number of replications that would be needed in order to be 95% confident you are within 5 minutes of the true mean patient wait time.

Rep	Base
1	50.8
2	63.8
3	51.2
4	98.2
5	123.5
6	63.8
7	47.8
8	33.5
9	43.0
10	112.5
11	59.5
12	52.7
13	109.6
14	95.2
15	21.8
Mean	68.5
Std dev.	31.3

8. A change is being proposed in the operation of the hospital's imaging department. The option was modeled and simulated for 15 replications. The mean patient wait time for the proposed option and the current system (base case) are shown in the table to the right. Is there a statistically significant difference in mean patient wait time between the current system and the proposed improvement?

Rep	Base	Option 1	Diff
1	50.8	93.3	-43.1
2	63.8	62.4	1.4
3	51.2	111.4	-60.2
4	98.2	51.5	46.6
5	123.5	29.3	94.2
6	63.8	37.6	26.2
7	47.8	60.7	-12.9
8	33.5	171.6	-88.1
9	43.0	85.5	-42.5
10	112.5	22.1	90.4
11	59.5	52.0	7.5
12	52.7	78.8	-25.1
13	109.6	61.0	48.6
14	95.2	35.6	59.7
15	21.8	60.5	-38.6
Mean	68.5	64.3	4.2
std. Dev.	31.3	29.2	55.4

Chapter

11

Including Reliability in a Simulation

As systems become more complex, additional functionality is often required for a more accurate representation of operations. Reliability has a significant impact on the performance metrics for a system. It also plays an important role in determining the location and size of any material accumulation points, which is critical for establishing an efficient production system.

Section 11-1 Performance

Analyzing a system's performance in the face of random events is often the goal of a simulation. *Performance* can be a confusing term as the parameters that define it may change based on historical practice or other factors. The following are commonly used measures of performance and are often used interchangeably.

- *Maximum rate*: Best possible output rate based on design
- *Ideal rate*: Maximum output rate based on theory
- *Target rate*: Output rate that is considered consistently achievable
- *Planning rate*: Output rate used for planning and scheduling purposes
- *Efficiency*: Actual output rate measured against some specified metric (usually target output rate)
- *Performance*: Some metric indicating how well a system is operating
 - *System Performance*: Based on an entire operating line
 - *Unit Performance*: Based on a single piece of equipment
 - *Plant Performance*: Based on multiple systems

Performance measures can take many forms.

Understanding how performance is measured and calculated is important because economic decisions will be made based on the defined measures. In practice, performance measures are often made as part of an assessment for an operation and can impact a financial bonus or result in a plant being closed. It is therefore essential that performance metrics in a simulation reflect the assumptions and calculation methods of the target system.

Simulation software provides many standard performance measures which are assessed at the object level. These include:

- *Throughput*: Number of items moving in and out of an object
- *Content*: Minimum, maximum, and average number of items in an object over the duration of a simulation
- *Staytime*: Minimum, maximum, and average time items spend in an object over the duration of a simulation

It is sometimes necessary to consider measures that describe system performance in ways other than the standard or pre-defined measures provided by the software. In those cases, custom code must be used in order to include the performance measures in the model.

Performance does not always have to be a measure of output or time. For example, in some industries, sequence is an important measure. In such cases, sequence measures how closely a set of products maintain their order from when they enter a system to when they exit. This measure is important in just-in-time systems.

Typically in an analysis, multiple measures of performance are considered and have to be traded off against each other in order to make a decision—of course the number of measures being considered should be limited to the most important ones. As the saying goes, we manage what we know and we know what we measure. We need to measure the critical few rather than the trivial many.

Section 11-2 Reliability

A major contributor to a system's performance is the reliability of the system's components. In this chapter, reliability is discussed from the perspective of equipment. While it's easier to think of stoppages and breakdowns in terms of manufacturing machines, the same concepts hold for almost any system. For example, a simulation of a call center includes "breakdowns" when the person typing information from the caller makes a mistake and has to go back and correct it. In a similar way, a simulation of a bank contains stoppages when a teller makes a mistake and has to call a manager over to help correct it.

The problem with reliability is that, in general practice, it is a term that is often misunderstood or incorrectly used. It's often applied interchangeably to entire systems and to single machines. Also, data collection time to estimate reliability may

Equipment reliability is a major factor influencing performance.

be too short. For example, an engineer may go to an equipment vendor and watch a test of a machine for a short period of time and declare its reliability to be, say, 98%, a percentage that is not appropriate for the collection time.

Common terms for reliability include the following:

- *Availability*: Percent of the time that a machine or system operates as expected
- *Reliability*: The probability that equipment can perform continuously, without failure, for a specified interval of time when operating under stated conditions
- *Mission length*: The interval of time that the equipment or system is expected to perform; the mission length may be years for a space probe or may be the length of a production run
- *MTBF*: Mean time between failures, the average time between failure/stoppage occurrences
- *MTTR*: Mean time to repair, the average repair time for the failure/stoppage

Reliability at a given time, $R(t)$, can be expressed as

$$R(t) = \Pr\{T > t\} = \int_t^{\infty} f(x)dx,$$

where $f(x)$ is the failure probability distribution. If the failure distribution is assumed to be negative exponential, then the failure rate is assumed to be constant over time. The constant failure rate is based on the exponential distribution's "memoryless" property and does not reflect burn-in time (when the failure rate decreases) or wear-out time (when the failure rate increases).

In the exponential case,

$$R(t) = e^{-\lambda t}$$

where λ is the constant failure rate.

Understanding the time involved is not as simple as it looks. The length of time indicated is usually not continuous clock time but is the time consistent with the activities associated with the failure process. This is called the *failure clock*. For example, a wrapping machine may jam while wrapping a product, so the time between failures is based on the time that the machine is actually wrapping. If the input flow is stopped for some reason, the time between failures should not accumulate. In a simulation, if the failure clock is not correctly maintained, a machine may show a simulated wrapping failure when the machine is actually sitting empty with no product in it; therefore, the failure clock in the wrapping example should only track the time the machine is in the wrapping state.

As shown in Figure 11.1 below, the downtime is the total time it takes to repair the failure and get the machine working again. Downtime includes the time for an operator to be aware of the failure, the time for the operator to complete a current task in order to become free to work on the failure repair, the time to travel to the machine, the time to actually repair the failure (including getting any tools and materials), and the time to start the machine back up.

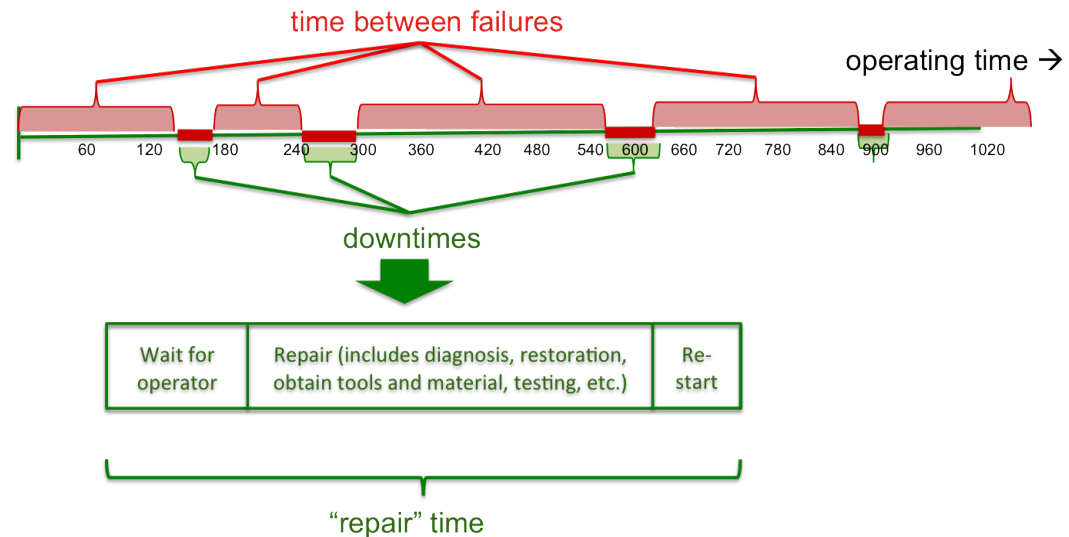


Figure 11.1 Operating and repair time periods

The term downtime and repair time should be defined consistently.

Oftentimes when operating personnel mention repair time, they refer just to the time needed to actually fix a machine; however, others may consider repair time as all of the activities and wait times involved from the time a machine fails until it is restarted. Repair time can therefore refer to two very different sets of activities and resulting times. It would be better if the term downtime were used to refer to the total time a machine is down due to a failure event, and repair time was only the portion of that total that involves hands-on repair; however, typically in a simulation

- without operators (or other shared resources used for repair), the repair time is the total downtime, as defined above;
- if operator (or other shared resource) intervention and travel is simulated, then the repair time is only the time to actually perform the repair.

Section 11-3 Estimating MTBF and MTTR

The time between failures and time to repair are usually random times that can be characterized by a statistical distribution. The MTBF (mean time between failures) and MTTR (mean time to repair) are the mean values of those distributions. As with any statistical analysis, obtaining historical data over a longer period of time is always

better; however, because of a lack of understanding about statistics in general, and reliability in particular, most organizations do not collect sufficient data.

A common metric for production lines is the total downtime during some period of time—a shift, a day, a run, etc. It's a metric that managers focus on and want to minimize; however, if only the total downtime is known, trying to focus on improving production is very difficult since the metric gives little detail for analysis. Plants will often keep records of the number of failures as well as the total downtime and use this to calculate an average downtime. Unfortunately, these metrics still don't offer much more toward a good analysis.

Modern manufacturing systems have the ability to track downtime information for individual machines. While this feature is helpful, there can still be problems. Providing only an average downtime along with the number of stops doesn't give any information about the underlying statistical distributions. Additionally, machines may stop for many reasons other than because of a failure. The machine may be blocked because of a bottleneck in a downstream machine or stopped for a reason related to production.

Exclusive downtime tracking software is available in many control systems. It not only tracks stoppages, but also indicates which machine is at fault as well as the nature of the fault. With such historical data, MTBF and MTTR can be calculated with some degree of certainty.

All is not lost if historical information is lacking or not available. For a new system with no history, a simulation can still be valuable for conducting a sensitivity analysis by experimenting with and analyzing various levels of MTBF and MTTR. Such analysis can lead to improved design decisions and provide reliability targets for new machines.

When simulating current operations with limited historical data, there are still some ways to estimate MTBF and MTTR. One way is to question operating personnel. However, simply asking for an operator's estimate of availability (or, in their opinion, reliability) may yield values that provide little useful information, such as stating the machine is available 95% of the time. It is better to ask specific questions, like the number of times a machine fails per hour (or any time period) and how long it takes to get it running again. It may then be necessary to ask additional questions to refine these estimates. Additional questions could help determine what activities the repair time values include, such as response time estimates, hands-on repair time, etc.

By knowing the number of downs per unit of time (in this case 1 hour) and the average time to repair, the MBTF and MTTR can be estimated since,

$$(MTBF + MTTR) * N = \text{Total operating time } (T),$$

$$MBTF = (T - N * MTTR) / N,$$

where N is the number of stops in time T.

There are many ways to estimate the time between failures and the repair time.

Such a simple estimate may have to be modified based on the specific piece of equipment and the failure clock time that should be used. Estimating the average downtime and repair time is only part of the work. The underlying statistical distributions also have to be estimated. The exponential and Weibull distributions are common starting points for downtime and repair time.

Section 11-4 Simulating machine failures

A single machine can fail in many ways depending on its activity.

The level of detail used to simulate a system should be consistent with the reliability data available. If the available data is only for a system of machines as a whole, then the simulation could be simplified by using a single machine with downtime characteristics to represent the system. Trying to simulate each machine is difficult, although it is often done by assigning the system failure characteristics only to the first processor.

In some situations, a simulation may need specific information for individual pieces of equipment. For example, a case packer machine will have separate failure modes for different steps or processes, such as

- collecting materials to pack;
- a case building section;
- the filling process;
- a case closure section;
- an output transport section.

Additionally, machines may have different types of failure modes: short-term (jams), intermediate-term (in-feed or out-feed conveyor problems), and long-term (belt failure). These multiple failure modes are called competing failures. Each competing failure will have its own statistical distribution and may have its own failure clock.

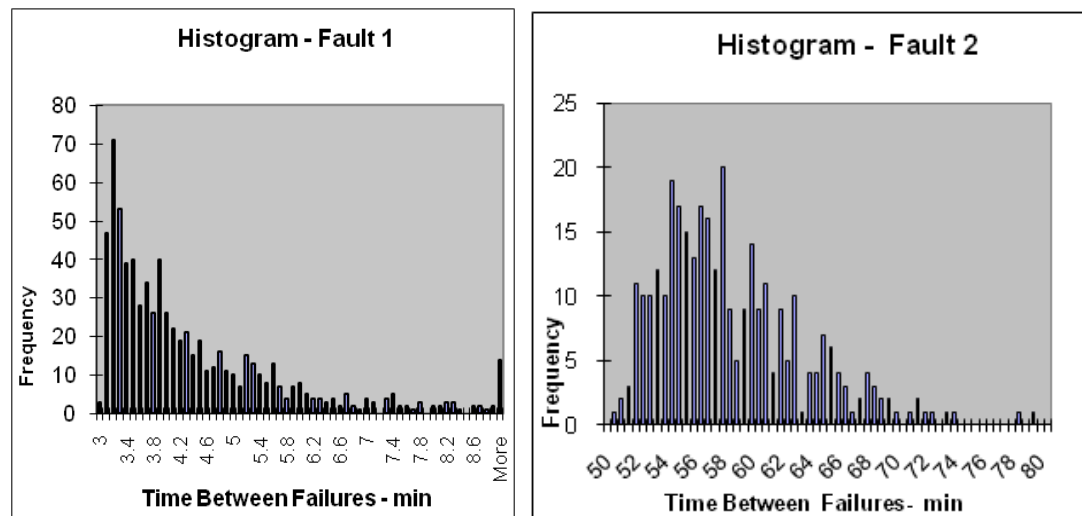


Figure 11.2 Competing failures

Without exclusive downtime data, identifying and characterizing competing failures is nearly impossible. Figure 11.2 shows the independent histograms of two simulated competing failures—the first with an MTBF of 4 minutes and the second with an MTBF of 60 minutes.

The combined histogram of the times between failures shown in Figure 11.3 demonstrates the problem that arises when the data are combined with no indication of the individual components.

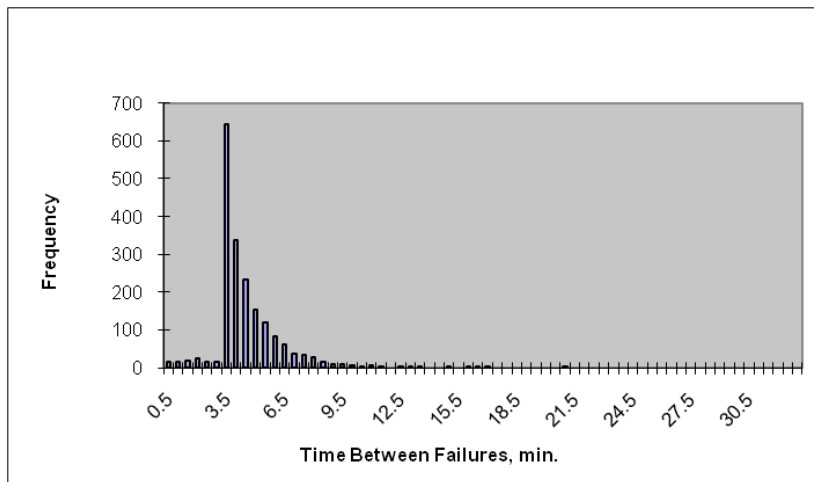


Figure 11.3 Combined failure histogram

Quite often all failures of a machine are grouped together and are then characterized by a single distribution and the resulting MTBF. Similarly, a single repair distribution is considered along with its MTTR. For machine design, however, the competing failures in different sections of the machine need to be considered separately since they both involve serial and parallel components.

Outside influences can also cause downtime; such downtimes may include operator break times, scheduled meeting times, and even preventive maintenance downtimes. Taking competing downtimes into account and correctly incorporating them into a simulation is important for conducting an accurate analysis. Many simulation programs work with only one failure clock, which is the same as the simulation clock. *FlexSim* allows for unlimited competing failure modes, each based on its own separate failure clock time; this is discussed in the next section.

Four types of information are needed to specify each type of downtime:

1. The object the specified downtime applies to (multiple objects can use the same downtime specification)
2. Distributions of time between failures and repair time
3. Actions that result when a downtime occurs (e.g., call an operator to make the repair)

Failures which occur based on different aspects of a machine are called competing failures.

4. The states that are used in the downtime or failure clock (for example, for a stoppage that only depends on operational or process time, the times the object is in the idle or wait for operator states are not counted).

Section 11-5 Setting MTBF and MTTR in a simulation

All simulation software applications provide some method for simulating random downtime events. The best way to understand how failures work in a simulation is to experiment with a small simulation model. The simulation, like the one shown in Figure 11.4, just includes a source, processor, and sink. To make checking the results easier, set the source interarrival time as well as the processor process time to a constant value of 1.

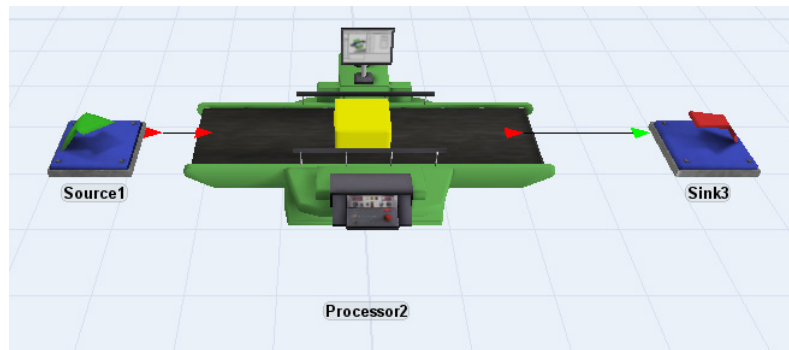


Figure 11.4 Using a simple simulation to understand how failures work

Most simulation software provides a means for entering downtime information.

In *FlexSim*, downtime information can be entered on the objects themselves by using the Breakdown tab on the object, as shown in Figure 11.5. To create the table, select Add.

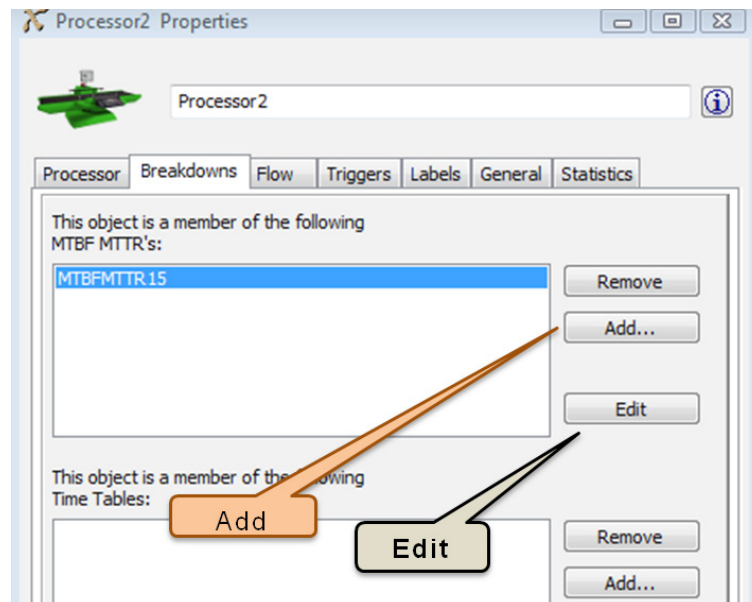


Figure 11.5 Object's Breakdown tab

Each table created represents a particular type of downtime that can affect that object or many objects. A single object can have multiple entries representing various downtime events. The tables created are stored as global tables and can be accessed through the Tools drop-down menu as well.

Selecting the Edit button on the Breakdowns tab after adding the table brings up the MTBF/MTTR Parameters Window as shown in Figure 11.6. The table parameters window has three tabbed sections that provide a very extensive and robust means for defining a particular reliability event such as what should take place when the machine fails or when it starts back up after repair. In the Members tab, objects that use this downtime table can be specified. They can all respond to the same event simultaneously or individually using the same base statistical distribution. The Functions tab, shown selected in the figure, is used to describe the statistical distributions associated with the breakdown. The Breakdowns tab is used to indicate what object states should be considered for this particular event. Any state or combination of states can be specified. Details for filling out the reliability table are provided in the Appendix.

The MTBF/MTTR tables contain all the information for a particular failure type.

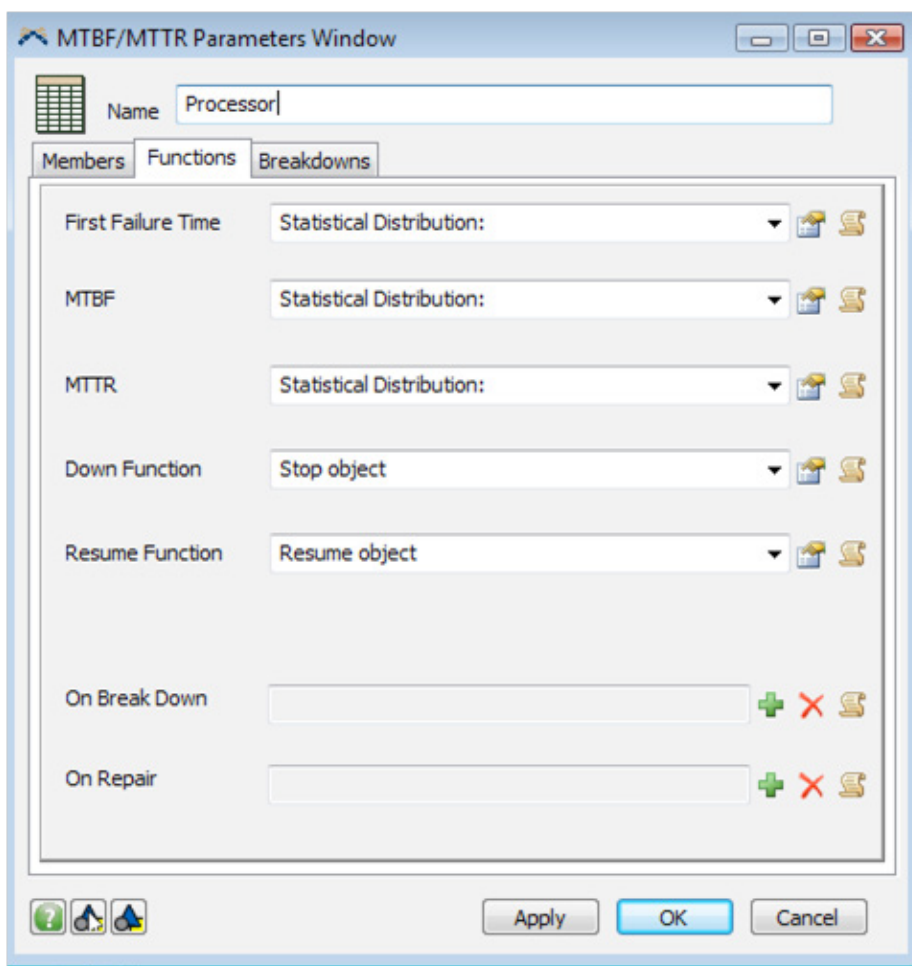


Figure 11.6 MTBF/ MTTR Parameters Window

A quick experiment will demonstrate why assigning states is important for reliability simulations. In the simple model defined above with an MTBF/MTTR table assigned to the processor, disconnect the connection from the source to the processor. Click the Reset button and then the Run button. After a period of time using the default values for MTBF and MTTR, the processor will have a yellow box around it showing that it is in a breakdown state and being repaired. This is illustrated in Figure 11.7. It is important to note that even though no material is going through the processor, it still shuts down periodically. This can be embarrassing when showing the simulation to a high-level manager or to others whom you are trying to convince that the simulation is accurate.

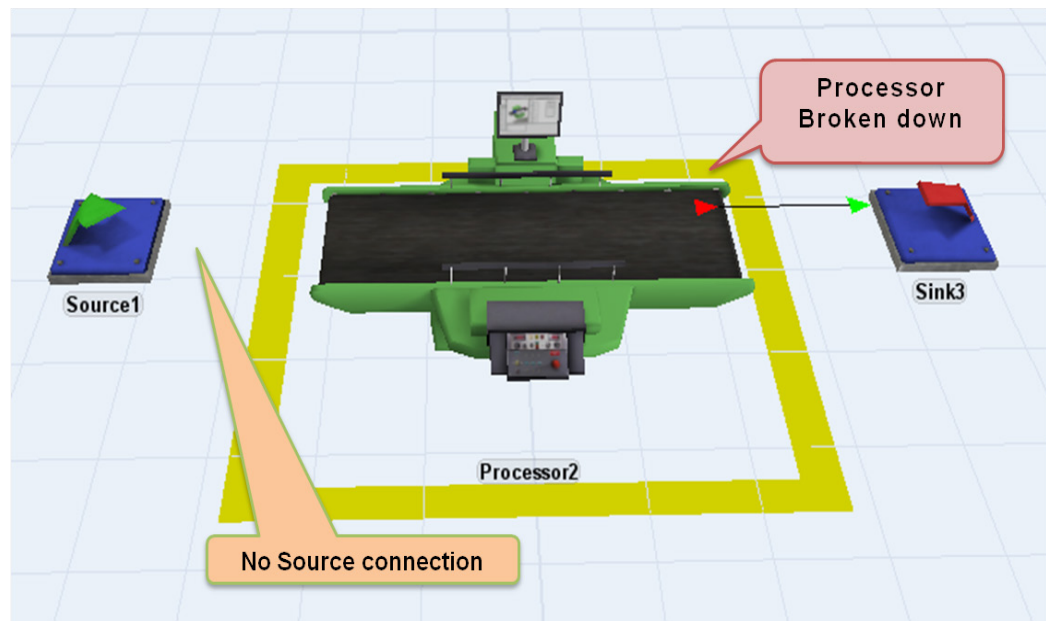


Figure 11.7 Processor failure occurs while not operating

A yellow box around an object indicates when the object is in a failed state.

This problem occurs because the failure clock, as discussed above, is the same as the simulation clock. In this case, the failure clock should only accumulate time toward failure when the processor is actually processing. To correct this problem, open the MTBF/MTTR table and click on the Breakdowns tab. As shown in Figure 11.8, check the box “Apply MTBF to a set of states” and then move the appropriate state(s) to the right column.

Repeat the experiment and notice that the processor no longer fails when it is empty. For an object such as a combiner, the failure time might be set up to apply to the collecting state as well as the processing state.

Try other experiments such as adding another processor in series or parallel. They can be given the same downtime characteristics (but not fail at the same time) by opening the MTBF/MTTR table, going to the Members tab, and bringing the new processor to the right-hand column.

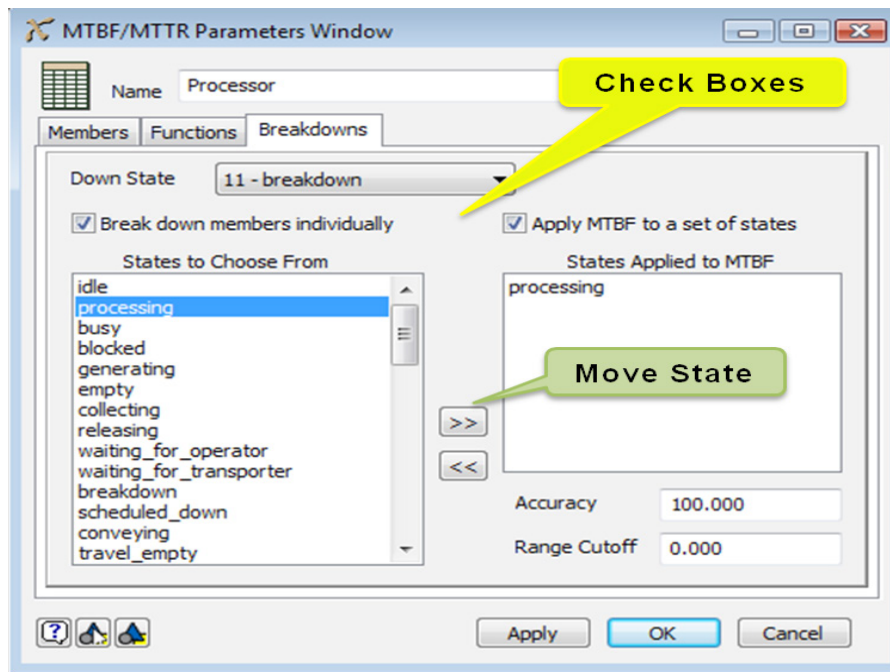


Figure 11.8 Setting the basis for the failure clock

Competing failures are added by creating a new MTBF/MTTR entry with the failure information and then choosing the same processor. A combiner may have one failure mode based on the collecting state and another based on the processing state.

Being able to accurately simulate reliability for so many cases means that data has to be collected, analyzed, and applied correctly.

Section 11-6 Using personnel for repairs

When dealing with downtimes, it is often necessary to simulate both the downtime itself and the response time of someone coming to perform the repair. To designate that another resource must be present to make the repair, choose the drop-down menu choice in the Down Function that will call an operator, as shown in Figure 11.9. All items in blue can be modified.

When it is time for an object in the list under the Members tab of an MTBF/MTTR table to break down, a call will be sent to the operator (or dispatcher) connected to the first center port of the down object. This is the default connection and can be changed if needed. If only one person can service the machine, that operator would be the one connected to the object with a center connection. Often the repair may be made by anyone in a pool of specialists. In the case that the repair can be made by any specialist the down object can be connected to a dispatcher that has several people (operators) attached to it.

The Preempt value can be important. There are two variables (the Preempt Value and the Priority value) that can influence what happens when an operator is called to perform a repair by default, task executors (e.g., operators, cranes, elevators, etc.)

The specific states of an object that can lead to a failure are identified on the breakdowns tab.

Personnel can be called to service a machine.

always finish a task before starting another. If the operator being called for a repair should stop its current task and go immediately to the repair task, the Preempt value should be set to 1. That means that when the repair task is complete, the operator will be freed to return to its interrupted task.

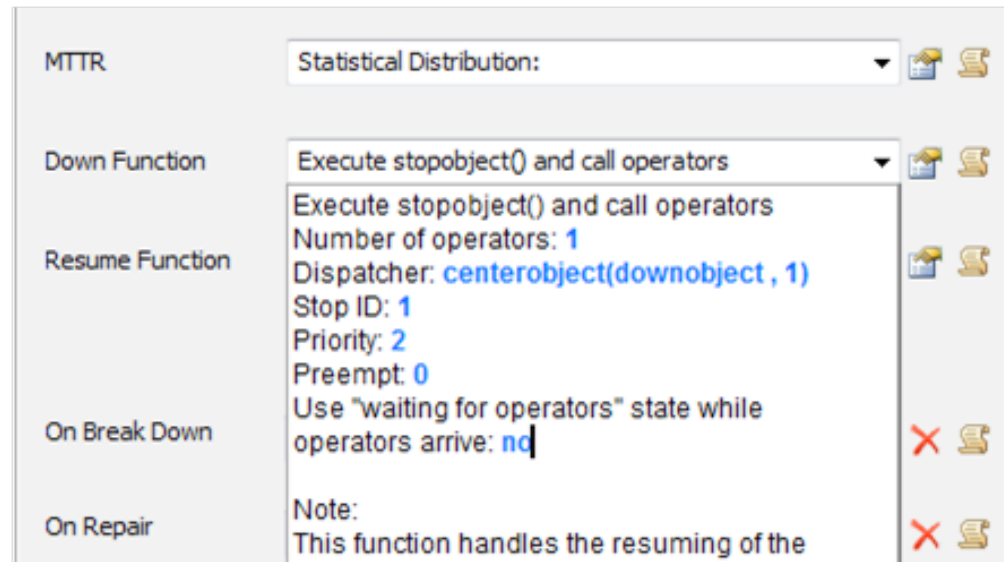


Figure 11.9 Setting the downtime action

A preempt value of 2 will abort the operator's current task and will not return the operator to that task. A value of 3 aborts all tasks the operator has to do. Values of 2 and 3 are only used with advanced programming. Since this particular Down Function automatically restarts the down object and frees the repair operator, the Resume Function should be set to Do Nothing.

The Priority value is used to decide which task of many the operator should perform next—the higher the number, the higher the priority. A Stop ID can be specified in order to trace the particular downtime event. If a “yes” is selected for “waiting for operators,” the time it takes an operator to respond to a breakdown will be added to that state for the down object.

While this discussion of interrupting the work an operator is doing is focused on responding to a repair task, the concept applies to all task executors, including transports, cranes, elevators, etc. This functionality provides a very robust way of simulating nearly any work rule environment where priority-based decisions have to be made in allocating resources.

Section 11-7 Surge

Surges are areas where material is stored while awaiting processing. Common surges include a raw material holding area or a warehouse where finished products wait to be shipped. A general principal of lean manufacturing is to minimized the use of surges as there are no value-added functions being performed. However, there are places where surges are useful.

Providing for some level of surge can be a means of protecting equipment from downtimes.

One method for improving throughput when machines incur failure modes is to use a surge in front of the processor. Although lean principles would argue against any such surge, a carefully designed surge can improve operations; however, there are some points to consider.

First, for any surge to be effective, the rate of the processor and all machines downstream of the processor (downstream resources, or DSR) must be greater than the rate upstream or those coming into the processor (upstream resources, USR). In other words,

$$\text{Downstream rate (DSR)} > \text{Upstream rate (USR)}.$$

If this condition is not met, the surge would simply fill each time the processor stops and never empty out. Additionally, the surge needs a *recovery time* (RT), which is the time it takes to empty the surge while the processor is running and material is coming into it. Obviously, to be effective, the surge should empty before the processor fails again. If, over time, that doesn't happen, then the surge will continue to fill until it is full and then can no longer be used as a surge. That requirement can be represented by the following equations

$$RT = MTTR / (DSR/USR - 1)$$

$$\text{and } RT < MTBF.$$

Try the following to experiment with surge in a simulation:

1. Build a simple model with a source, conveyor, surge, processor, and sink to experiment with surge size and the ratio of downstream and upstream rates.
2. Start with a source rate that is less than the processor rate.
3. Create an MTBF/MTTR table for the processor with processing as the failure clock.

The requirements for surge size are only approximate because of the statistical nature of the failure modes. Experiment with different values of surge size and ratios of source and processor rates.

Exercise 11-1 Kegglers Brew

Background

Kegglers has found a niche market by selling their microbrew beers only in keg containers. Their main customer base consists of local taverns and restaurants. During an economic downturn they have found that demand for their product actually goes up. As director of marketing you have recognized that there may



be a growing market for smaller sized kegs, so management made the decision to add two smaller sizes: A quarter keg and a Fridge Keg. Focus groups have been very enthusiastic about the new sizes.

The company has decided to update the current line so that it is capable of producing all three sizes of kegs. Their current line of standard kegs runs with an operating efficiency in the high 90s. Thinking there may be some inefficiencies associated with producing the smaller sizes, the business plan is based on an 85 percent operating efficiency.

Problem statement

Should the plant manager agree to the target numbers?

Operating data

As shown in the Figure 11.10 and the accompanying data table, empty kegs are taken from storage and put on a conveyor by a placement machine. The kegs arrive in the filling room by conveyor and go through the filling machine. From the filling room the kegs move on a conveyor to the packaging area where they are placed on pallets. Plastic sheeting is wrapped around the kegs and pallet to keep them together. After wrapping, the pallets are conveyed to a pickup queue (with a capacity of 2) where a fork truck gets the pallet and transfers it to cold storage.

Product	Keg arrivals Kegs/min	Filler Rate Kegs/min	Kegs/ Pallet	Pallet Cycle Time (sec)	Max Pallets Per 8hr Shift
Full Kegs	10	12	6	23	800
Quarter Kegs	16	18	8	28	960
Fridge Kegs	24	30	12	30	960

Table 11.1 Basic equipment data

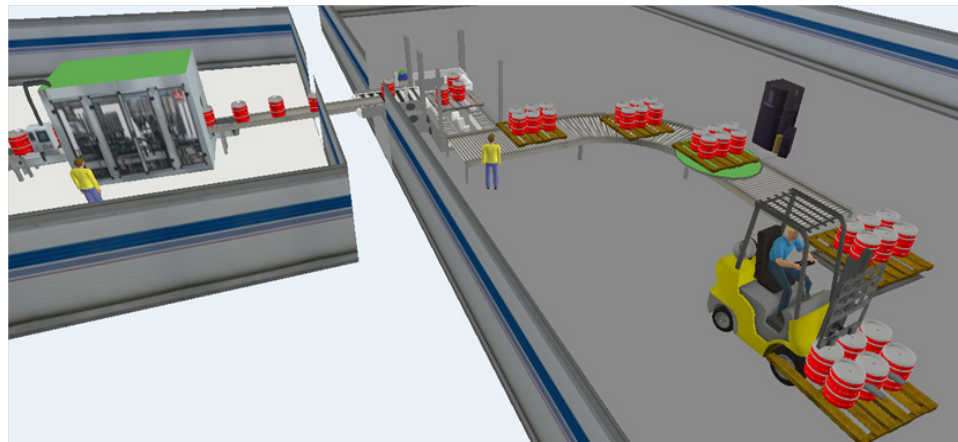


Figure 11.10 Keg line from filler through wrapping

The wrapper takes 30 seconds regardless of the number of kegs on the pallet. The fork lift, which carries one pallet at a time, takes 15 seconds for a one-way trip between the pallet pickup queue and the warehouse.

The keg size is constant at 0.4, 0.4, 0.6 grid units.

Conveyor	Length – ft	Capacity	Speed – ft/sec
Source to Filler	10	40	0.2
Filler to Palletizer	10	20	0.5
Palletizer to Wrapper	10	4	0.25
Wrapper to Pickup	5	5	0.5

Table 11.2 Conveyor sizes and speeds

Reliability information, taken from historical downtime data of similar equipment, shows the following times and distributions.

Equipment	MTBF(sec)	Distribution	MTTR(sec)	Distribution
Filler	900	Exponential	90	Uniform: min 60; max 120
Palletizer	1000	Exponential	90	Uniform: min 60; max 120
Wrapper	1200	Exponential	120	Uniform: min 80; max 160

Table 11.3 Times and distributions

Expected results

- Draw an OFD for the system.
- Run the simulation for a number of 8-hour shifts (28,800 seconds), showing the number of pallets brought to the warehouse during that time; compare the results to the maximum and the expected total.

Modeling and analysis issues

- What time units should be used?
- What would be the advantages/disadvantages to the following ways to simulate the line operation?
 - A separate simulation for each product?
 - One simulation changing variables for each product run?
- For the machines with changing rates, where should the data be stored?
- How can you change the number of kegs on a pallet easily?
- How would you validate that the model to determine if it is working correctly?
- What inter-arrival rate should be used on the keg source? The same question applies to the pallet source for the combiner.

- The keg size to be used is given in the problem statement.
 - Where should the size command be set?
 - If a different size is used for each product, how might it affect the results?
- What visualization improvements could be made?
- How can the fork truck travel time be set?
- In this problem both sources have to have the item type set correctly for the same product. How can the Experimenter be used to change item types?
- Where is the bottleneck? Will the use of a queue be helpful and how can it be easily added to the simulation?

Details of the simulation are located in the Appendix.

Exercise 11-2 Chairs for Tots

Background Chairs-for-Tots builds children's chairs that are sold through major outlets and over the internet. The production facility is just keeping up with their order rate but is also trying to keep operating costs down. Work in process at the end of the week is continued the next week. Ideally, operators help clean the area at the end of the week. However, if there are orders still to be completed, the cleaning operation becomes difficult. The plant doesn't want to add overtime but management feels they have to do something to increase the weekly output and still have time to clean and service the equipment.



Problem statement

What will be the impact of higher order rates on the operations?

Operating data

The predicted order rate for chairs is expected to be approximately one every 30 minutes (distributed exponentially). The plant starts operations at 8 a.m. Monday morning and runs 24 hours a day. It stops taking orders at noon on Friday so that the work in progress can be completed, the area cleaned, and the workers sent home at 4 p.m.

There are 4 major production steps:

- For each order, a set of raw materials is conveyed to the cutting machine loading queue. An operator moves the wood from the queue to the cutting machine, which operates automatically.

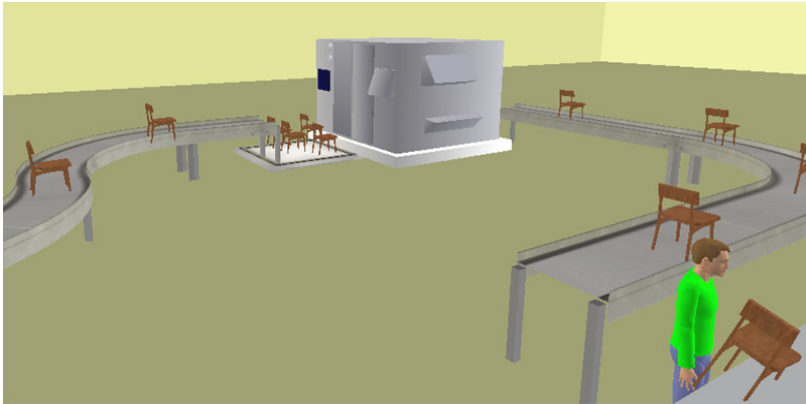


Figure 11.11 Drying and packing areas

- Once cut, the cutting operator places the set of parts on a conveyor. The parts move to another queue where they wait to be assembled and painted. The assembly operator takes the parts and then assembles and paints each chair.
- The painted chairs are conveyed to an oven to dry and set the glaze. The oven can hold up to five orders. Therefore, five orders are usually accumulated and placed in the oven at one time. However, any order shouldn't wait more than 150 minutes to be placed in the oven.
- When dry, the orders are transported by conveyor to the packing area where they are inspected and prepared for shipment.

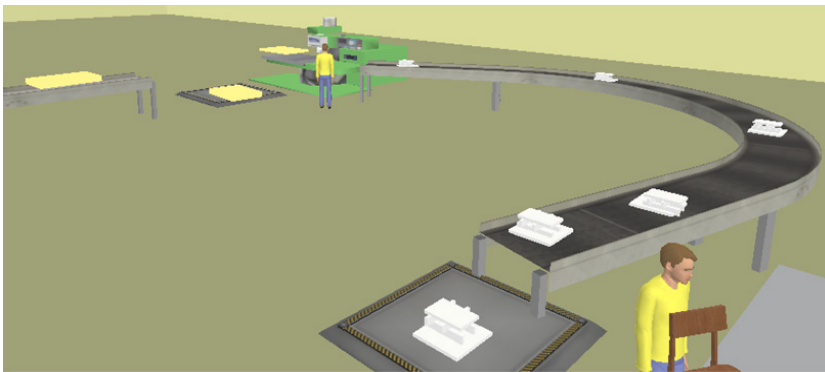


Figure 11.12 Cutting and assembling areas

Operating times (minutes)—the varying times reflect the size distribution of the orders.

Operation	Attribute - Minutes	Distribution
Order arrival rate	Mean = 30	Exponential
Cutting	Min 10; Max 20	Uniform
Assembly	Min 10; Max 20	Uniform
Heat Treatment	60	Constant
Inspection	Mean 20; Std 5	Normal

Table 11.4 Operating times

Downtime occurs in a number of ways. The cutter and assembly operation both have frequent and infrequent downtimes. The frequent downtimes can be repaired by the operator. The operator will always stop the present job to repair the machine. The infrequent downtimes (changing a cutter blade and repairing the paint sprayer) have to be handled by the plant mechanic, who responds to breakdowns as they occur. The plant mechanic also has to fix problems in the other plant areas.

Operation	Time between failures (minutes)	Distribution	Time to repair (minutes)	Distribution	Repair Person
Cutter	Min. 20; Max. 80	Uniform	Min. 1; Max. 4	Uniform	Cutter Operator
Assembly	Min. 15; Max. 25	Uniform	Min. 0.8; Max. 1.4	Uniform	Assembly Operator
Oven	Mean 420; Std.Dev. 20	Normal	Mean 5; Std. Dev. 2	Normal	Mechanic
Cutter blade change	760	Constant	Min. 10; Max. 15	Uniform	Mechanic
Paint sprayer	Min. 400 Max 560	Uniform	5	Constant	Mechanic
Other plant areas	Mean 100	Exponential	Min. 10 Std. Dev. 5	Normal	Mechanic

Figure 11.13 Historical records for downtimes (minutes)

Material travel times

- Order entry to cutter: 3 minutes
- Cutter to assembly: 3 minutes
- Assembly to oven: 3 minutes
- Oven to packing: 5 minutes

Capacities

- Conveyor to cutter: 10
- Queue before cutter: 10
- Conveyor to assembly: 10
- Queue before assembly: 5
- Conveyor to oven: 10
- Conveyor to packing: 10

Expected results

- Draw an OFD for the system.
- Run the simulation for 10 workweeks. Although in practice any incomplete orders would carry over to the next week, consider each week a separate run starting with a clean line.
- Report on production levels as well as how many orders are still in the system at the end of the week. The only operating personnel who need to be considered are the cutter operator, assembly operator, and plant mechanic.

Modeling and analysis issues

- What level of detail is needed? Should all chairs in an order and their parts be tracked?
- What should be the basic time unit?
- How can you start and stop the simulation?
- Although the combiner object can perform batching is it a good choice here?
 - What other object can handle batching?
 - How can the max waiting time be handled?
- What is the best way to validate that the components are working correctly before adding the complexity of the downtime?
- How can the “rest of the plant” be simulated and given a downtime for the mechanic to fix?
 - Which downtimes will pre-empt other jobs?
 - How is the person to service the downtime selected?
- How can the Experimenter be used?
- What performance measures are important?
- What can be done to improve operations so that all jobs are finished by 4 p.m. on Friday?
 - The union wants an additional mechanic added.
 - The cutter operator wants an automatic infeed from the cutter queue.
 - The lean team suggests changing the way orders are entered into the oven—even suggesting converting to a five-lane continuous oven.

Chapter 11 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Describe three different performance measures, such as those listed in Section 11-1, including one that is not based on a rate measure, and give an example of each as applied to an operations system.
3. Discuss how does reliability can impact system performance.
4. Discuss the difference between “availability” and “reliability.”
5. When including reliability in a simulation model, it is important to understand the definition of the word time in mean time between failures. Discuss why this is important in a simulation model.
6. Discuss the different times that can be referred to in mean time to repair.
7. Describe the different ways you can estimate MTBF and MTTR.
8. Explain what competing failures are and how they can be simulated.
9. Assume the reliability of a machine is given by

$$R(t) = e^{-0.01t}, \text{ where } \lambda = 0.01$$

that is, the failure rate is 1 per 100 hours.

- a. Describe in words the meaning of $R(t)$.
- b. What is the probability that the machine will run 200 hours before failing?
- c. For this case does the answer change whether the previous failure occurred 10 hours ago or 300 hours ago?
10. Describe the two ways to enter reliability data in *FlexSim*. How would you set MTBF and MTTR for three machines that share the same distributions?
11. What does the appearance of a gold box around a piece of equipment signify?
12. Describe what happens when the Preempt box is checked for an operator who is being used to repair more than one piece of equipment.
13. Explain the basic concepts of using a surge to protect from down time.

The Advanced User

Chapter

12

Customizing Model Logic

While the Intermediate User can study a wide variety of dynamic systems using simulation, there will still be a need to simulate actions that cannot be described using the preset logic. The Advanced User has enough understanding of the simulation software to create custom logic and use advanced features to analyze more complex situations. Depending on the ease of accomplishing these tasks, the Advanced User may not need to be a simulation or programming professional.

Most Advanced Users will have either a definite interest in simulating systems or will carry out simulations on a relatively frequent basis. An Advanced User normally will also be knowledgeable of the subject matter that makes up the simulation problem. While this section develops an extended knowledge of the simulation software, it does not completely describe all of the advanced functionality available in the application.

The primary background required for the Advanced User is knowledge of the underlying software structure and the basic programming commands for building custom logic. The Advanced User also becomes more proficient in dealing with information that is exchanged between a simulation and external programs. Additional skills include utilizing the charting and reporting functionality of the software, enhancing the visualization of the model, etc. This section provides an overview of the software architecture and its scripting commands. More information is provided in the *FlexSim* Users' Manual, available through the Help menu.

Section 12-1 Hierarchical software architecture

The *FlexSim* software is unique in a number of ways from the other simulation applications discussed in Chapter 2. The most important is that it has an object-oriented software structure. Object-oriented programming was developed to improve

Advanced users can handle more complex applications but may not necessarily be an expert in the simulation language.

functionality and maintain software quality as computer hardware and software became increasingly complex. An object-oriented program is a collection of interacting objects. Each object can be viewed as an independent entity with a distinct role or responsibility and which knows how to manipulate its own data structure. This is in contrast to the approach that had been dominant for many years that keeps data and functions separate and considers the application program as simply a long list of commands and tanks to be performed.

The use of object-oriented techniques in the development of *FlexSim* means that it is hierarchical and data oriented in nature and consequently can be visualized by a tree structure used for databases. It's called a tree even though the visualization is generally upside down compared to an actual tree, with the "root" at the top and "leaves" at the bottom. A structured representation of a simple simulation containing four elements is shown as a tree in Figure 12.1. A tree structure is conceptual, and appears in several forms. Common examples of tree representations are the folder/file graphical interfaces found on computer systems, e.g. *Windows Explorer*. In *FlexSim*, the basic elements in its tree view are called nodes.

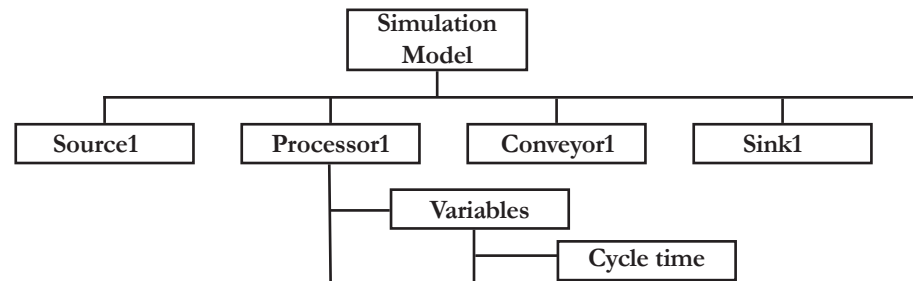


Figure 12.1 General hierarchical object tree structure

The tree provides a very robust, visual, and efficient way to find where variables are stored or where particular sections of code reside. It is a modern alternative to the traditional way of programming applications. Older software applications were created by writing code as a long list of commands. Code that was reused in the application was established as a sub-routine that could be recalled for use; however, searching through all of the lines of code and subroutines to find a reference to a variable, or knowing where certain commands are located, was confusing and difficult. The tree, with its hierarchical, object-oriented structure changes all that.

The following list contains the primary attributes of the tree:

- *Tree groupings*
 - *Main tree:* Application structure and project-related objects and data
 - *View tree:* GUI related objects and picklists
 - *Model tree:* Model related objects and data, part of the main tree
- *Nodes:* the basic building blocks of the tree; they

The tree is a visualization tool for the software and its hierarchical structure.

- hold all the model definition information “behind the scenes”;
- have a name;
- can have a data item associated with it: a number, string, or object.

Consider a simple model made up of a source, queue, processor, conveyor, and sink, as shown in Figure 12.2. To see the tree view of this model in *FlexSim*, right click in a blank space on the surface and select Explore Tree from the popup menu. The tree structure appears with the model as the first reference. Holding and moving the left mouse button scrolls through the tree, as does the scroll bar on the right. Figure 12.3 shows the tree view for the model in Figure 12.2.

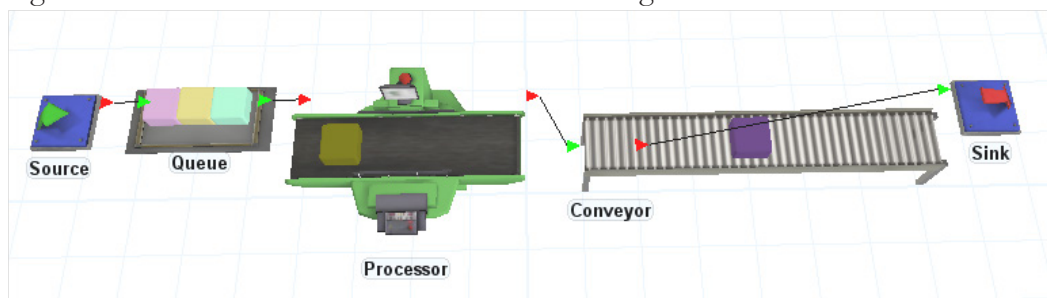


Figure 12.2 Simple simulation model

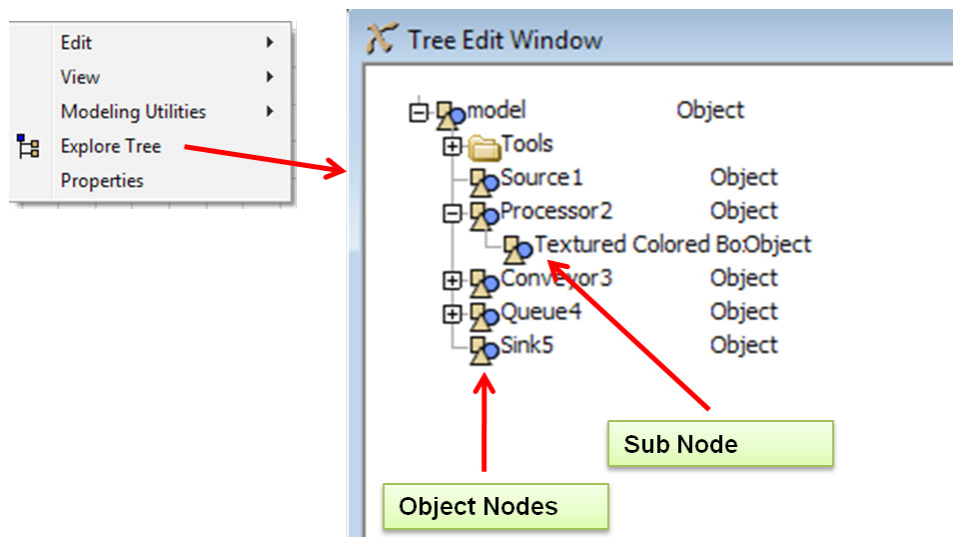



Figure 12.3 Tree structure window

The nodes in the tree show a clear path to all the variables and logic in the model.

The nodes in the *FlexSim* tree hold all of the information that a *FlexSim* model uses to run a simulation. The *FlexSim* tree stores all the parameters and properties for objects, graphical user interfaces, event lists, and other information. The nodes representing the objects in the model are called *object nodes*. Object nodes contain data nodes and functions that define how the object looks and behaves. Note that to the left of the object data node, there is a control that looks like a plus sign in a box. This control means that the node has subnodes, which can be viewed by clicking on the

icon. The subnodes represent objects, such as flowitems, that are contained within the primary object node.

The node names generally represent the type of data within them. For example, the *class* and *superclass* nodes contain information about the parent objects from which the selected object inherits its behavior. The *visual* and *special* nodes contain data about the appearance and location of the object. The *labels* node contains the data on the object's labels.

Many of the data nodes can be expanded into subnodes by clicking to the left of the node. As shown in Figure 12.4, clicking on the Processor2 node results in the dive icon  appearing to the left of the object node icon. Pressing the dive icon expands the node to show the nodes containing the processor's data.

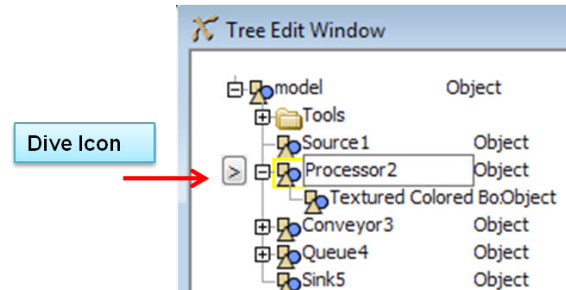


Figure 12.4 Dive icon

The variables node for an object contains all the important variables that affect its operation.

The variables node is especially important because it contains most of the operating variables and logic for the object. An expansion of the variables node is shown in Figure 12.5. Nodes like *cycletime* contain the code chosen for that item on the interface. If these nodes contain custom code that is to be executed, then the circle on the node changes to a square with an S in the middle, indicating that the code is *Flexscript*. Values of nodes can be changed by using *Flexscript* commands and providing a path to the node from any location. The path can take the form of a typical file structure path or be defined automatically through the use of simplified *Flexscript* commands. These commands are discussed in Section 3 of this chapter.

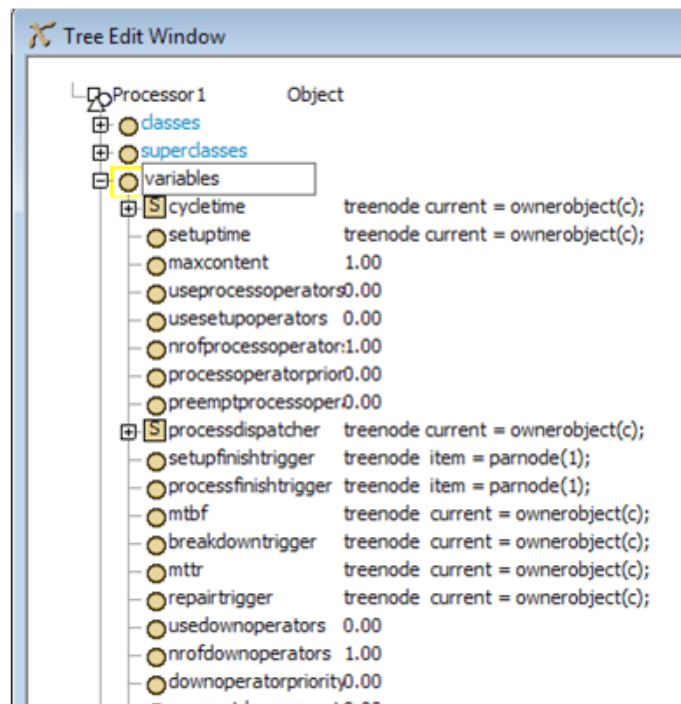


Figure 12.5 Expanding the variables data node

A summary of tree node symbols is provided in Figure 12.6; they indicate the functionality of the node.

With this important software structure, all nodes, variables, and other functions can be easily located and modified dynamically. Many of the nodes are directly linked to entries on the user interface of individual objects. That means that the value is dynamically synchronized in both locations.

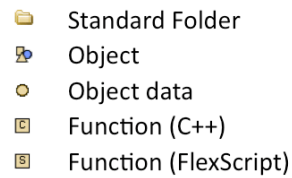


Figure 12.6 Tree node symbols

Section 12-2 Understanding how objects work

The previous chapters of this book used various characteristics of objects without really discussing how they worked in detail. This section examines what happens to an object when a flowitem enters it.

The Figure 12.7 shows the order of events for a processor, but it is similar for other object types. The first thing to happen when an item enters an object is the execution of the OnEntry trigger. Its function is to provide a means of customizing what an object does when a flowitem enters.

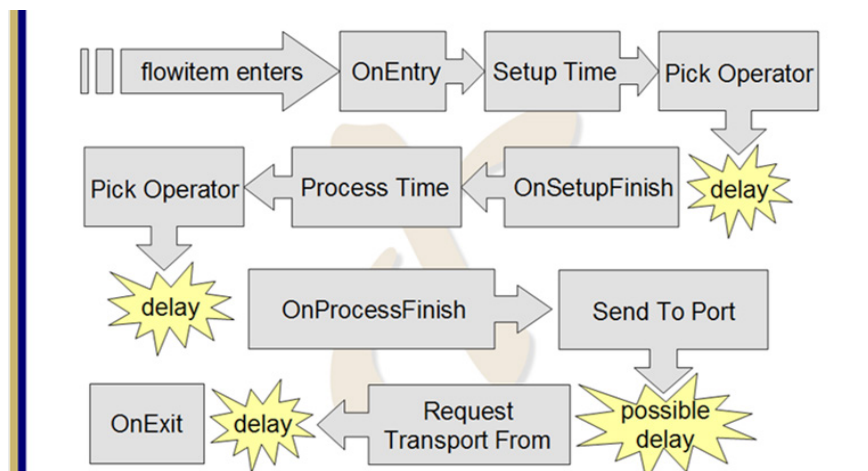


Figure 12.7 Order of event execution (pushed flowitem)

For the processor's next step a variable called Setup Time is evaluated. The setup time is a mathematical expression that returns to the object how long to delay the flowitem in order to represent a setup time. If a value of zero is returned, then the delay time is zero. This execution is different from a trigger in that it is always executed; triggers are only executed if there is logic associated with its designated event.

If an operator (task executer object) is required for use during a setup time, the Pick Operator field is evaluated. This field returns a reference (usually via a center port connection) to a task executer object. The next step is a delay that is equal to the value returned from the evaluation of the setup time trigger, plus any time required by the task executer to become available and to travel to the object.

There is an order of execution for all events in an object; knowing that order can often help explain behavior.

Upon the completion of the delay, a trigger called `OnSetupFinish` fires to signal the end of setup time. This event, as with all such events, allows the modeler to add customized behavior.

The step marked `Process Time` behaves the same as setup time. The `Process Time` field is evaluated, and the value returned to the object represents the length of time needed to process the flowitem. Next an operator is picked, if applicable.

Another delay is incurred which corresponds to the delay given by process time plus any travel time for the selected task executor. The delay is followed by the firing of the `OnProcessFinished` trigger. This trigger signals the end of the process time and may be used to add logic that should occur before a determination is made as to where to send the item. Such logic may start a downstream machine to get it ready to accept the item.

Once the process time is complete, the `SendtoPort` field is evaluated, which returns a value to the object that signals what output port to send the flowitem through. The step marked in Figure 12.7 as “possible delay” is important because it will effectively stall execution of the object until any delay happening downstream of the object is over. The possible delay is caused by the situation where the downstream object may not be ready to receive the flowitem—the downstream object may be at full capacity, busy processing, etc. It is a possible delay (as opposed to a definite delay) because it is purely a timing issue between the upstream and downstream objects. This delay happens more often than not in most models.

`Request Transport From` is the next field on the `Flow` tab to be evaluated. This field behaves like the `Pick Operator` fields for process and setup times. It returns a reference to a task executor that is assigned to come and transport the flowitem, followed by a delay. If a task executor is requested by this field, the delay corresponds to the time it would take the task executor to finish its current task (if applicable) and travel to the object making the request. This isn’t listed as a possible delay, because the travel time to the object will always take some amount of time.

Lastly, the `OnExit` trigger fires just before the flowitem exits the object. When the flowitem causes this trigger to fire, the port and operator (if applicable) are already known and engaged in transporting the flowitem away. Neither port nor operator can be changed at this point, as the flowitem is already gone, as far as the object is concerned.

Because of this order of execution, the `OnExit` trigger of a `Source` is an ideal place to assign itemtypes to the model’s flowitems. The `OnCreation` trigger is the other choice, but generally speaking, it is safer to assign the itemtypes during `OnExit` since it is the last possible chance to make a change to a flowitem before it is in a different object.

A common misconception is to confuse the functionality of the `SendtoPort` field with the `OnExit` trigger. These occur at different times and function in different ways. The `SendtoPort` field is a basic function of the object and always occurs as shown in the figure. The port number that it returns is set. The `OnExit` trigger

occurs when the item actually leaves the object and, at that point, it cannot override the value returned by the `SendtoPort` function.

Section 12-3 Scripting basics

The picklist options, introduced for the Intermediate User, provide a wide range of commands to control the simulation without writing any code; however, if a simulation is intended to examine truly unique strategies that provide a definitive advantage, then writing some level of unique, custom code is to be expected.

FlexSim uses a scripting language called *Flexscript* to create custom logic, which is modeled around the syntax of the popular programming languages *C/C++* and follows most of the same conventions found in those languages. Anyone familiar with *C/C++* can quickly begin writing custom modeling code.

FlexSim also provides an alternative programming paradigm called the Logic Builder. The icon to the right of each logic pick list indicates how the underlying code is displayed. A script icon represents *Flexscript* while a puzzle piece represents the Logic Builder. To be sure that *Flexscript* is used to display the logic the checkbox “use Logic Builder by default” located at file/Global Preferences/Environment should not be checked. There is a toggle icon at the bottom of all code windows that allows switching between *Flexscript* and the Logic Builder.

Some basic rules and guidelines for using *Flexscript* include the following:

- *The language is case sensitive:* “A” does not equal “a”; `Processor1` is not the same as `processor1`.
- *No specific formatting is required:* Use of spaces, tabs and line returns is encouraged for more “readable” code.
- *Text strings are entered between quotes in order to be flagged as text:* For example, “`sometext`” would be treated as a string, `sometext` would not. It is important to note that the code error checker will NOT check the spelling or correctness of text fields.
- *Parentheses follow a function call, and commas separate the arguments of the function:* For example, `moveobject(object1, object2);`
- *A completed statement (function or command) must always end with a semicolon (;)*
- *Parentheses can also be used freely to make associations in math and logic statements:* For example, `(x+3)*2`. Parentheses must always be paired. If you put your cursor next to an opening parenthesis, *FlexSim* will highlight it and its closing partner. Putting your cursor next to a closing parenthesis will highlight it and its matching opening parenthesis.
- *Curly braces { } are used to define a block of statements:* Blocks of statements also define a scope and are used with `if`, `for`, and `switch` statements.

Scripting is the means to change logic and create custom, advanced functions.

***Flexscript* shares most programming characteristics of C++.**

- *To comment a line use // followed by the comment:* Comments appear in green text and are ignored by FlexSim. The comment area ends with a line return.
- *Multi-line comments start with /* and end with */.*
- *It is best not to use spaces or special characters in name definitions:* The underscore character (`_`) is the only “safe” special character. Think of it as a 27th letter of the alphabet.
- *Names may include numbers but may not begin with a number:* For example, `machine_9` is acceptable; `9machine` is not.

Flexscript–Variables

There are different types of variables. The type determines what kinds of values the variables can hold.

- *Integer*, declared **int**, can only hold integer values: the natural numbers, their negatives, and zero.
- *Double*, declared **double**, can hold the set of real numbers, such as 3.14159, -6.1, etc. The double type is considered a more precise variable type compared to the integer type: while a double type may hold an integer value, an integer type may not hold a real number.
- *String*, declared **string**, holds text which is specified within quotation marks; (e.g., `"my Text"`).
- *Treenode*, declared **treenode**, is unique compared to the other three variable types and only exists within FlexSim. Treenodes hold references to nodes in the FlexSim tree.

Before a variable can be used it must be created or declared. A declared variable must have its type and name given in the declaration, but no value need be assigned in a declaration. For example, the following would correctly declare a double variable “result” that would hold the result of a sample from a uniform distribution

```
double result;
result = uniform(0, 10);
or double result = uniform(0, 10);
```

The single line of code is equivalent to the two lines above. Other examples are

```
int index = 1;
double weight = 175.8;
string category = "groceries";
treenode MyProcessor = node("/processor2", model());
```

Flexscript–Operators

Treenode is a variable type that is a pointer to a specific node in the tree.

Variables have to be declared before they can be used.

Operators are the mechanisms used to manipulate values and variables. Some of the operators will look familiar, such as the standard math operators, +, -, /, and *. A list of more advanced math operations that require the use of functions is found in Figure 12.8. For other math functions, see *FlexSim Help/Commands*.

<code>sqrt(x)</code>	returns the square root of x
<code>pow(x, y)</code>	returns x to the power of y (x^y)
<code>round(x)</code>	returns x rounded to the nearest integer
<code>frac(x)</code>	returns the decimal portion of x
<code>fmod(x, y)</code>	returns the remainder of x/y

Figure 12.8 Advanced math operations

In addition to the mathematical operators, there are operators for comparing values as shown in Figure 12.9. The results of these comparisons can be true (represented by the number 1) or false (represented by the number 0):

<code>></code>	Greater than. Used as $x > y$.
<code><</code>	Less than. Used as $x < y$.
<code>>=</code>	Greater than or equal to. Used as $x >= y$.
<code><=</code>	Less than or equal to. Used as $x <= y$.
<code>==</code>	Equal to. Used as $x == y$.
<code>!=</code>	Not equal. Used as $x != y$.

Figure 12.9 Operators for comparing values

It is important to note that the double equal sign of the equivalence operator is not to be used interchangeably with the single equal sign of the assignment operator. They mean very different things.

Another way to compare text is by the use of the function `comparetext(string1, string2)`; which compares strings of text for equivalence. Note that strings cannot use the operators given above for numbers. Other commands are used for working with strings. See *FlexSim Help/Commands*.

Three operators that are used for constructing logical relationships, which are useful for complex *if-else* decisions are shown in Figure 12.10.

<code>&&</code>	Logical AND operator. Used as $(1 \ \&\& \ 2)$. Used in a statement to return true if multiple conditions are met;
<code> </code>	Logical OR. Used as $(1 \ \ 2)$. Used in a statement to return a true if either condition is met;
<code>!</code>	Logical NOT. Used as $(! \ 0)$. Used in a statement to return true if any condition but the value given is met.

Figure 12.10 Operators used for constructing logical relationships

Flexscript—commands

Flexscript contains a large number of commands that provide easy access for referencing or getting information from particular nodes in the tree structure. They are

Flexscript commands are custom, high level instructions that carry out common software operations needed for simulation.

also used to perform other functions, such as interacting with *Excel*. The command choices under the Help menu provide definitions of all commands as well as examples of their use. Commands are arranged alphabetically and by primary function.

Figure 12.11 shows the expressions used for assigning or updating the value of a variable.

=	Assignment. Used as $x = y$. x is assigned the value of y .
+=	Add and Assign. Used as $x += y$. x is assigned the result of $x + y$.
-=	Subtract and assign. Used as $x -= y$. x is assigned the result of $x - y$.
*=	Multiply and Assign. Used as $x *= y$. x is assigned the result of $x * y$.
/=	Divide and Assign. Used as $x /= y$. x is assigned the result of x / y .
++	Increment. Used as $x++$. x is assigned the value of $x + 1$. Also written $x += 1$.
--	Decrement. Used as $x--$. x is assigned the value of $x - 1$. Also written $x -= 1$.

Figure 12.11 Expression for updating variables

Referencing objects

The tree structure of *FlexSim* gives objects hierarchical relationships to each other. These relationships allow for referencing objects quickly and easily using commands and keywords. Custom logic usually means interacting with other objects. For example, changing operations in one section of a simulation when a downstream section experiences a particular problem.

There are two important keywords when referencing within a single object:

- **current** references the present object—the one that owns that code.
- **item** references the flowitem currently in the object that caused the custom behavior to execute.

The involved flowitem is the flowitem that caused some event to happen. For example, when an OnEntry trigger fires, the flowitem that just entered the object is designated as **item**, even if there is an item already present in the object, as could be the case with a queue.

As noted above, an object node can be referenced through a path and through its connection to other objects. A node anywhere in the model can also be referenced by its rank within

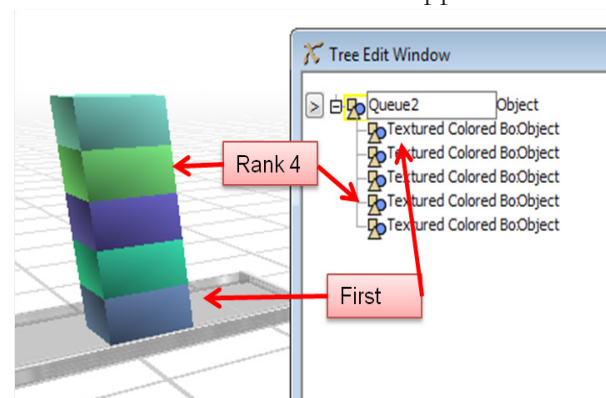


Figure 12.12 Identifying subnodes by rank

Nodes in the tree can be easily referenced in many ways.

its parent tree through a function, called `rank()`, that takes the container node (that is, the node it is a part of) as the first argument and the rank number as the second argument. For example, as shown in Figure 12.12, to get a reference to the fourth flowitem within a queue (the queue being the current container node in this case) and use it in a variable, use the rank function as follows:

```
treenode fourth = rank(current, 4);
```

An object node may also be referenced by its relative first or last position in a tree. To get a reference to the first subnode of a tree you can use the `first()` function, which takes the container node as the only argument. The `last()` function works the same way but returns a reference to the last node. For a queue with a FIFO default logic, `first(current)` would refer to the flowitem that is the next to leave.

Additional references to objects are made by specifying the object through port connections:

- `inobject(object, port);`
- `outobject(object, port);`
- `centerobject(object, port);`

For example, as shown in Figure 12.13, if you need to reference an object connected to the first input port of the current object, you would use

```
inobject(current, 1);
```

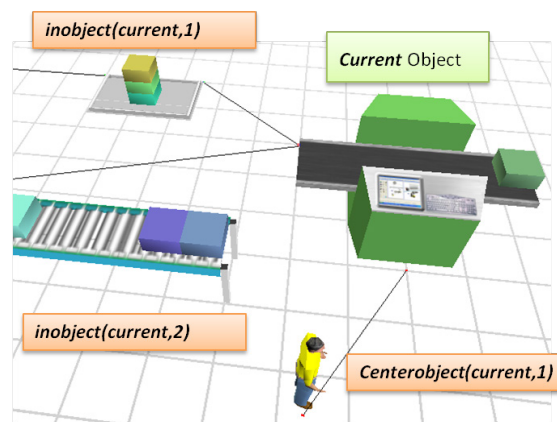


Figure 12.13 Referencing objects by port

A final command used to reference a node is the `node()` function, which takes a text string as its first parameter defining the path to the node. Its second argument is the node from which to begin searching. This command invokes a search for the node specified in the path string. It can be slow compared to the other commands discussed.

Consider a simple simulation with two processors that are connected in series, as shown in Figure 12.14. If simulation logic is needed to reference Processor2 from a trigger in Processor1, there are two ways to make the reference:

```
outobject(current, 1);
```

or

```
node("/Processor2",  
      model() );
```

Here, Processor2 is referenced from within Processor1. In the `outobject()` command case, Processor2 is described as the object that is connected to output

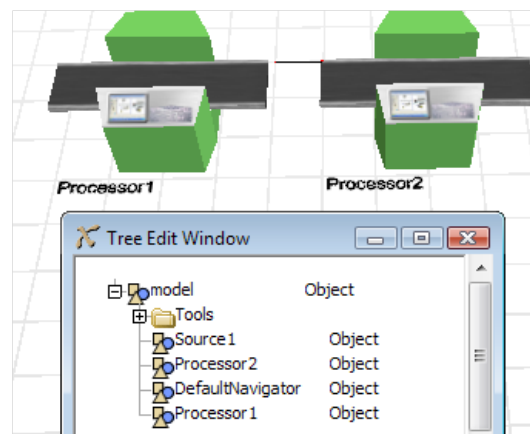


Figure 12.14 Referencing using the node command

port 1 of the current object (Processor1). This command, as written, is simpler but is only valid when created and used within Processor1. Although slower, the node command is valid anywhere in the simulation.

Flexscript–Other helpful functions

Flexscript commands provide direct ways to extract statistical data from the model:

- `content(object)` returns the number of items currently contained in the referenced object node. When used on a queue, for example, it will return the number of flowitems inside the queue at the time the `content()` call is made.
- `getinput(object)` returns the number of items received by the object.
- `getoutput(object)` returns the number of items released by the object.

Other commands get and set data in labels and tables. These have the format of

- `getlabelnum(object, "labelname")` get the numeric data value from the specified label on the specified object;
- `gettablenum(object, "tablename", row, column)` gets the numeric data value from the table contained in the row and column specified.

Data included as nodes in the variables node of an object can be obtained and set by commands such as

- `getvarnum(object, "maxcontent")` gets the numeric data value of the variable specified on the specified object.

In the last three commands the information specified surrounded by “ ” must be the text name for the label, table, or variable. The name must be spelled correctly and is case sensitive. For each “get” command there is usually a corresponding “set” command. See the command list in the *FlexSim* Help for a full list of *Flexscript* commands.

Section 12-4 Creating custom logic

Flexscript code can be added nearly anywhere within the simulation. For this text, however, most custom logic will be added in picklist options. The script window is found by clicking on the A button to the right of the picklist option.

In Figure 12.15 the Set Itemtype picklist option is selected on the OnExit trigger of a processor. The *Flexscript* code is modified as follows:

1. Bring up the scripting window by pressing the script icon (the A button to the right of the OnExit picklist). In this case it is the script associated

Flexscript functions provide quick access to common object information.

Not all logic can be accomplished using pre-built commands.

with setting the itemtype. At the top of the script window, three treenode variables are defined by default. These include the variables `item`, `current`, and `port`. The use of comment fields and other commands are needed for the code template view.

2. To see the code without the comment and template fields, and to prepare for adding or modifying the code, click the clear template view icon at the bottom of the window. The resulting window starts with a comment section containing the words *Custom code*. Replacing this string will change the text in the drop-down window that defines the logic. Comment sections that indicate the range of the pick option remain but can be deleted.
3. Clicking on the checkmark icon at the bottom of the screen checks the syntax of the code. Any errors will pop up in a compiler console window at the bottom of the *FlexSim* window. The console indicates the type of error and the line number. In this case, a semicolon has been omitted.

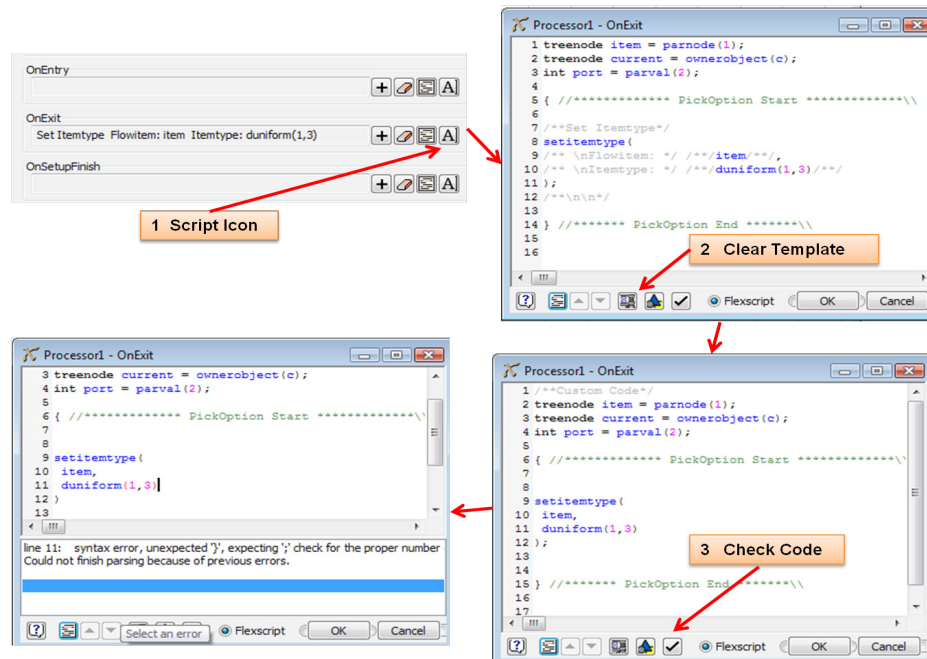


Figure 12.15 Script window characteristics

There are also visual cues to help keep the code correct. Notice that keywords and commands are shown in blue, and numerical values are shown in pink; data types and logic constructs are in bolded black. It is easy, therefore, to spot a potential error even before running the code checker because it will not appear in the appropriate color. (Default colors may be changed by selecting File/Global Preferences.) For example, misspelling the keyword “current” would result in:

curent rather than `current`

or

Current rather than `current`

Clearing the template code makes remaining logic easier to view.

Script syntax can be checked before closing the code window.

Color-coded syntax checking helps to catch errors quickly.

Comments
make script
easier to
understand.

It is important to realize that string data contained in quotes is not checked for accuracy. Therefore, any misspelling will result in the function not operating, but no error message will be given. For example, trying to read the maximum content variable of a queue coded as `getvarnum(current, "maxcotent")` (notice the missing 'n' in "maxcontent") would return a value of zero, but would not give an error message.

Use the `//` marks to create comments which can help in remembering what the logic is supposed to do. These marks indicate that anything typed from them to the right to the end of the line is a comment. Use `/*` to start and `*/` to end when commenting out a series of lines.

Some of the most common logic expressions involve either "**if**" statements or "**for**" loops, so it's critical to have the syntax correct. In each case, both parenthesis (), and brackets { }, are used to identify the scope of the script. Putting script on separate lines, noting comments, and using indents makes for easier reading and debugging. For example, consider the following if statement, for loop, and switch statement (note the use of the semicolons (;) used on all statements except the **if**, **else**, **for**, **switch**, and **case**):

If statement

```
/* If the current content of the current object
is >5, then do statements 1 & 2. Else do 3 & 4. */
if(content(current) > 5)
{
    Statement1;
    Statement2;
}
else
{
    Statement3;
    Statement4;
}
```

"If" statements,
"for" loops,
and "switch"
statements
are commonly
used script
structures.

If the braces { } are omitted after an 'if' statement, only the first line will be executed.

For loop

```
//Repeat statements 1 & 2 five times, increment x by 1
for(x=1; x<=5; x++)
{
    Statement 1;
    Statement 2;
}
```

Switch statement

```
//Take action based on the value of variable

switch(variable)
{
    //in the case that variable==1, execute statements
    1 & 2

    case 1:
        Statement1;
        Statement2;

        //the break statement takes control out of the
        switch statement

        break;

    //in the case that variable==2, execute statements
    3 & 4

    case 2:
        Statement3;
        Statement4;

        break;

    //default provides a place to go if variable is
    not a 1 or a 2

    default:
        break;
}
```

Checking for errors

When finished writing the custom code, it is important to click the check mark icon at the bottom of the edit window. If there are errors they will be noted along with the reason and the line number of the code where the error occurs. Most common errors include missing a parenthesis or bracket, missing a semicolon at the end of the line, misspelling a command, or misplacing a comma.

As previously stated, some errors are not found by the syntax checker but will yield an incorrect result if not corrected. These include misspelling a text string enclosed in quotes, misplacing a comma (in a command that doesn't give a syntax error but changes the command meaning), or providing an incorrect table reference.

**Not everything
is checked for
errors.**

Exit the code window correctly to save changes.

When finished checking the code, click on the OK button at the bottom of the window to save any changes. Clicking on Cancel or the X in the top right corner of the window will not save any changes.

Dividing by zero can cause unpredictable results, so to protect from dividing by a variable that may be zero because of an error or because of another reason (such as a count of items at the beginning of a simulation), test the value of the variable in the denominator before using that variable. Using the command `maxof(0.0001, variable)` in the denominator is a good way to accomplish this.

A list of *FlexSim* commands is provided in the Appendix. Description details and examples can be found in the Commands selection in the Help menu. Also in the Help menu is the *FlexSim* User Manual which contains a section on writing logic in *Flexscript*.

Global variables

A variable that is used many times in a simulation may be defined once as a global variable and then referenced whenever it is needed. Global variables are found in the Tools drop-down menu.

As shown in Figure 12.16, there are four steps involved in setting up a global variable:

1. Click the Add button on the Global Variables window.
2. Select a name for the variable.
3. Select the type of variable.
4. Set an initial value.

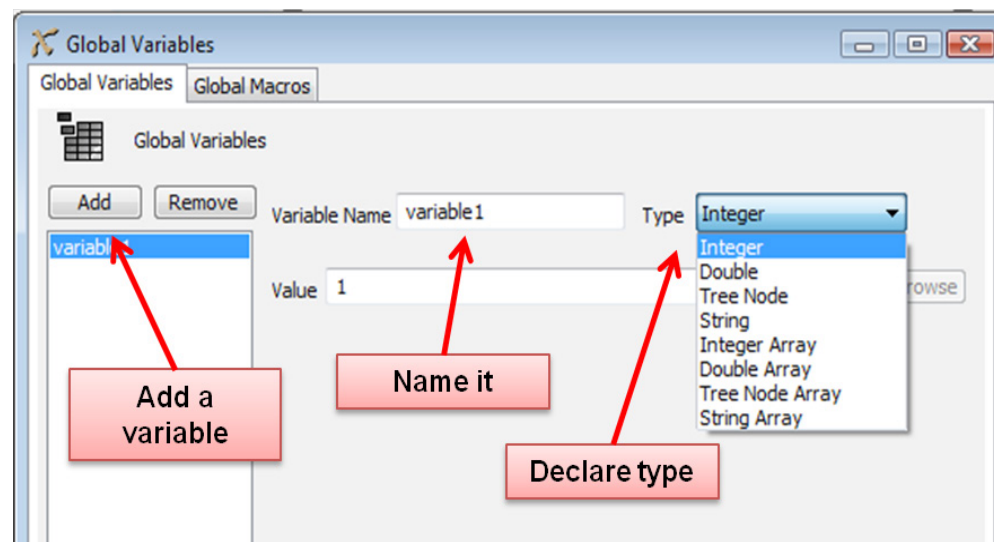


Figure 12.16 Creating a global variable

If the variable is a treenode type, use the browse button to select the node, and the path will be automatically assigned to the variable.

The initial value of the variable, as seen in the Global Variables menu, does not change dynamically. The actual value will remain at its last assigned value; therefore, it may be necessary to initialize the value with an OnReset trigger.

Once a global variable has been established it can be used in logic statements anywhere in the simulation simply by referencing it since it has already been created. For example, an integer variable declared as a global variable and named project can be used as

```
project = 7;
x = project + 2;
```

A global variable defined as a treenode that points to Processor1 can be referenced as `getoutput(Processor1);` // use the variable

Local and block variables

In addition to global variables, there are two other types of variables used in *FlexSim*: local and block. A variable is only valid within its scope—the portion of code where it is declared. Considering the example in Figure 12.17, “value” is a block variable since it is declared within the block of code defined between the braces. The other variable, “counter” is a local variable because it is local to the code window (e.g., one of an object’s triggers, process time, send to logic). In this case, the statement `return value;` would not be valid since value is not recognized outside of its scope, the code block.

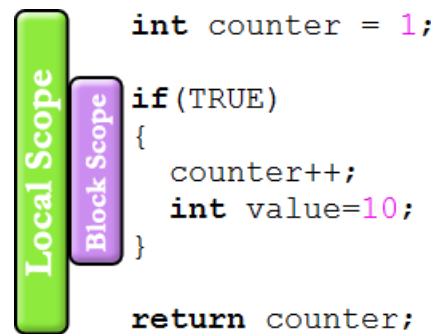


Figure 12.17 Local and block variable scope

Troubleshooting hints

There will be times when the simulation does not seem to work correctly even though the syntax checker does not indicate any errors. If the logic checker indicates that the behavior is unexpected, it may not indicate a real error in syntax. There is also the possibility that there may not be a problem, because, while the behavior is unexpected, it may be correct based on how the logic is implemented. In either case, it is important to find the logic that’s causing the problem. There are a number of techniques and tools available to help troubleshoot.

One approach is to print out information at various points in the logic either to a file or to the Output Console, which is opened by selecting it from the View menu at the top of the screen. The following commands print to the console:

- `pt("text string")` prints a text message.
- `pf(number)` prints a number as a floating point value.

Print statements help with troubleshooting or better understanding logic.

- `pd(number)` prints a number as an integer value.
- `pr()` with no parameters prints the next statement on a new line.

If each of the above commands are preceded with an “f”, `fpt("text string")` for example, the result will be printed to a file. The file must be opened and closed with the *Flexscript* commands `fileopen("filepath")` and `fileclose()`. Only one text file can be open at a time.

It is a very common and useful practice to use several `pt()` and `pf()` calls together to track the progress of a value during model execution in order to make sure things are happening as expected. For example the following code,

```
pt("Item in "); pt(getname(current)); pt(" "); pf(time()); pr();
```

would print to the output console

- the text within the quotes;
- the name of the current object;
- an empty space;
- the current model time;
- a print return for a new line.

Closing and re-opening the output console clears its content.

In addition to print statements, the `stop()` command can be inserted into the logic to stop the simulation when it reaches a particular point in the logic. The simulation can be continued by clicking on the Step icon or the Run icon.

The simulation can also be stopped by setting a stop time using the stop time field option located on the simulation control panel near the top of the screen next to the simulation time indicator. With the stop time set, the simulation can be run at higher speeds until it reaches the point of interest.

Section 12-5 Custom logic example

This section provides a simple example that illustrates some of the referencing and coding concepts described in the chapter. The base model is the simple queueing system shown in Figure 12.2. Specific model parameters include:

- Time between arrivals \sim exponential(0,10,1)
- Service time \sim exponential(0,9,2)
- Time between server failures \sim exponential(0,100,3)
- Time to repair server \sim normal(5,1,3)

The analyst would like to include the following model enhancements:

1. Calculate and display, as the model runs, the average time items spend in the system (time between entry and exit).
2. Server works 10% faster when there are at least four items in the queue.

As with any modeling or coding activity, there are multiple ways to approach the problem; therefore, the method described here is just one of several ways the enhancements can be implemented.

The first enhancement implements the following formula:

It is implemented on the OnEntry trigger of the sink object. The pseudocode that is applied is:

- Determine the current item's time in system (TIS) by subtracting the time the item arrived into the system from the current time.
- Update the total time in the system for all items so far (TotalTIS) by adding the current item's time in system to the previous total; use a global variable.

Notes:

- Define the global variable named TotalTIS in the Tools>Global Variables menu; it is of type double and has an initial value of 0.
- Reset the value to 0 for each run.
- Get the total number of items leaving the system (TotalItems)
- Calculate the average time in the system (AvgTIS), the total time in the system (TotalTIS)/total items exiting (TotalItems)
- Write the values to a global table: number of items exiting (TotalItems) and average time in system (AvgTIS); reset the values to 0 for each run.

$$\bar{X} = \frac{\sum_{i=1}^n X_i}{n} = \text{AvgTIS} = \frac{\sum_i TIS_i}{i} = \frac{\text{TotalTIS}}{\text{TotalItems}}$$

- Display the average time in the system on the modeling surface via a visual tool.

The *Flexscript* code that implements the pseudocode is shown in Figure 12.18. It is implemented in the OnEntry trigger of source object named Sink. The code utilizes several *Flexscript* commands: `time()` returns the current time, i.e. the current value of the simulation clock; `getcreationtime(item)` returns the time the current flowitem was created; `getinput(current)` returns the number of flowitems that have entered the current object, in this case, the sink; `settablenum("gt_Output", 2, 1, AvgTIS)` sets row 2, column 1 of the global table named `gt_Output` to the current value of AvgTIS, the average time in the system. The `pf()`, `pd()`, and `pr()` commands are used to verify that the code is working as intended by writing the current values of the variables to the Output Console as the model runs.

Once verified, this line can be deleted or commented out. By just commenting out the commands, the code can be easily verified again later if the logic is changed.

Since the output is written to a global table, the table `gt_Output` is defined with the number of items exiting (`TotalItems`) stored in the first row, first column and the average time in system (`AvgTIS`) stored in the second row, first column.

Since a global variable is used to store the total time in the system, it must be defined and reset to 0 for each run. This is done through the `OnReset` trigger in the sink; this one line of initialization code is also shown in Figure 12.18.

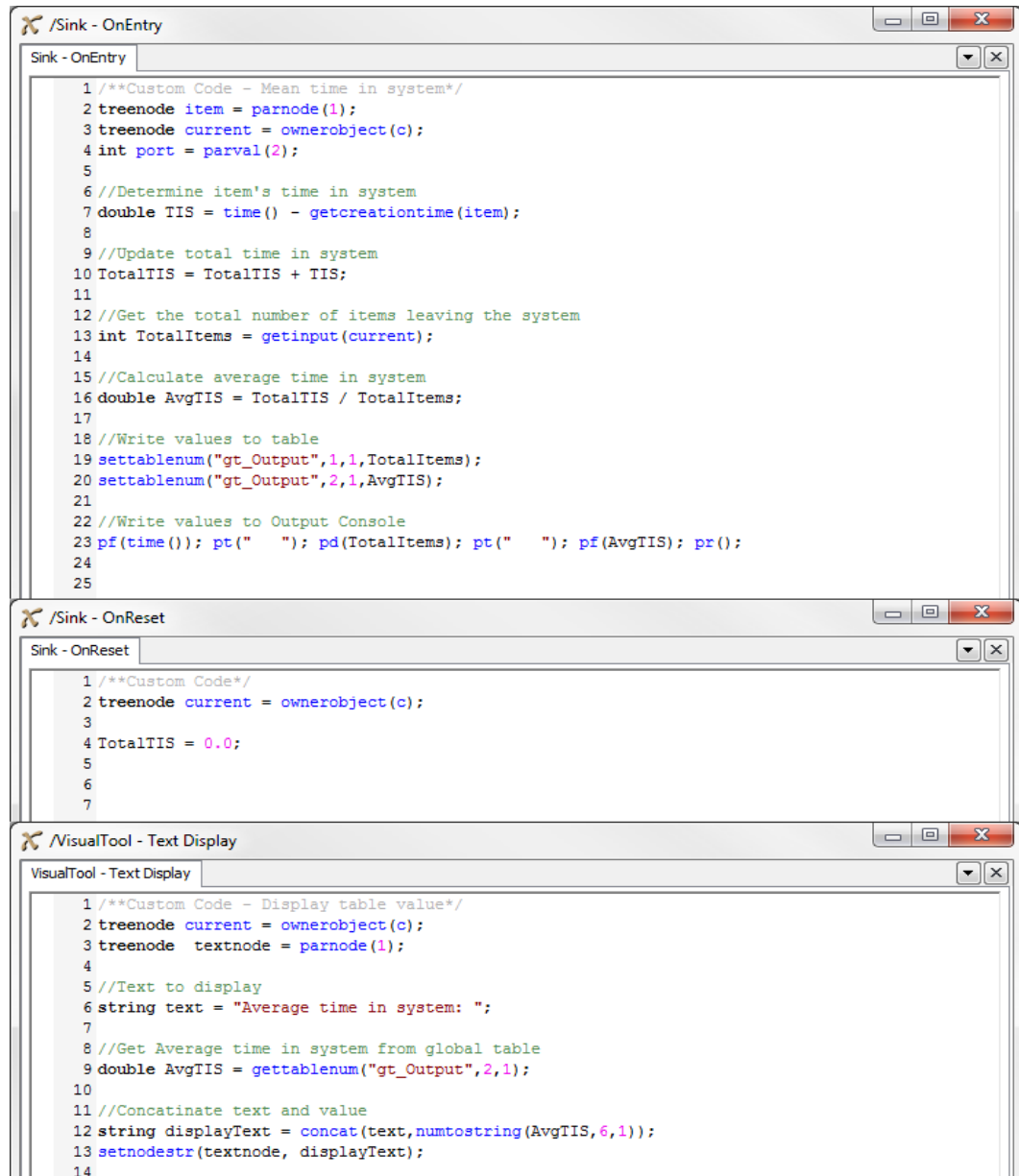


Figure 12.18 Code to determine and display average time in the system

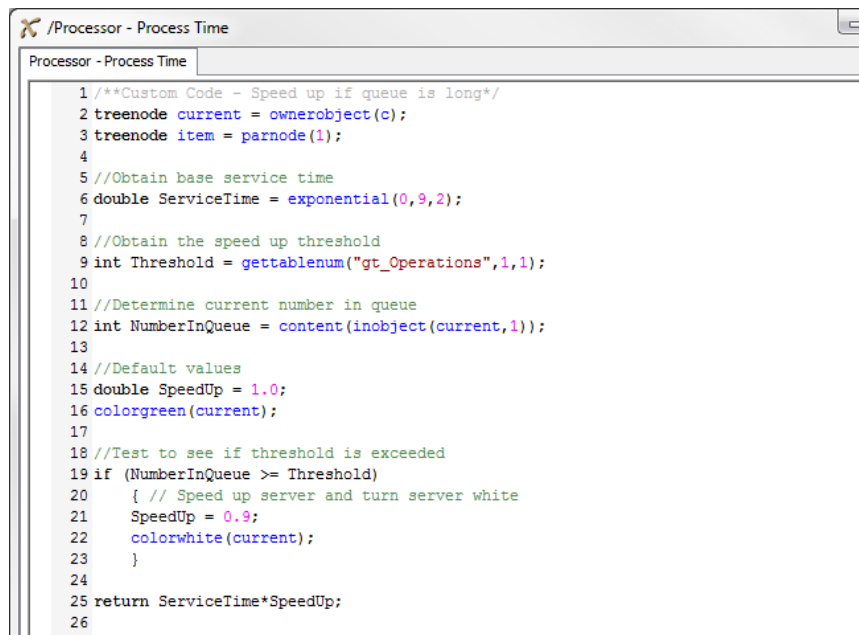
A visual tool is used to display the average time in the system (from the global table) as the model runs. The code, as shown in Figure 12.18, is

implemented in the Text Display dropdown. The code utilizes the *Flexscript* command `gettablenum("gt_Output",2,1)` to get the average time in system from row 2, column 1 of the global table named `gt_Output`. Since the visual tool is used in text mode, the average time in system value must be converted from a number to a string through the *Flexscript* command `numtostring(AvgTIS,6,1)`, where 6 is the minimum number of characters (numbers and decimal point) to display and 1 is the precision or number of decimal places displayed. The converted value is then concatenated with the descriptive text through the command `concat(...)`. Finally, the `setnodestr(textnode,displaytext)` command attaches the information that is to be displayed (contained in the string variable `displaytext`) to the object node (defined by the `treenode` variable `textnode`).

The second enhancement to the basic model, where the server works 10% faster when there are at least four items in the queue, is implemented in the Process Time trigger of the processor object. The pseudocode that is applied is:

- Obtain the base service time by randomly sampling from the service time distribution.
- Obtain the threshold value for working faster (stored in the global table `gt_Operations`).
- Determine the number of items currently in the queue when service starts (OnEntry to the processor).
- Test to see if the current number in the queue exceeds the specified threshold.
- If the current number in the queue exceeds the threshold, then reduce the base service time by 10% and turn the server white.

The *Flexscript* code that implements the pseudocode is shown in Figure 12.19.



```

1 /**Custom Code - Speed up if queue is long*/
2 treenode current = ownerobject(c);
3 treenode item = parnode(1);
4
5 //Obtain base service time
6 double ServiceTime = exponential(0,9,2);
7
8 //Obtain the speed up threshold
9 int Threshold = gettablenum("gt_Operations",1,1);
10
11 //Determine current number in queue
12 int NumberInQueue = content(inobject(current,1));
13
14 //Default values
15 double SpeedUp = 1.0;
16 colorgreen(current);
17
18 //Test to see if threshold is exceeded
19 if (NumberInQueue >= Threshold)
20 { // Speed up server and turn server white
21   SpeedUp = 0.9;
22   colorwhite(current);
23 }
24
25 return ServiceTime*SpeedUp;
26
--

```

Figure 12.19 Code to speed up server if queue is long

The code utilizes several *Flexscript* commands. The `gettablenum("gt_Operations", 1, 1)` command obtains the threshold value for deciding whether or not to speed up the process time which is stored in row 1, column 1 of the global table named `gt_Operations`. The nesting or combining of two commands `content(inobject(current, 1))` is used to get the current contents of the queue through `content(...)` and the reference command `inobject(...)` that points to the object that is connected to the first input port of the current object, which in this case is the queue. The code also uses `colorwhite(current)` and `colorgreen(current)` to change the color of the processor (current object) depending on whether it is in speed-up mode or not.

Note that every item that enters the processor has its service time set by sampling from the exponential distribution. Also, the speed up value is set to 1.0 and the color of the processor is set to green. If the contents of the queue meets or exceeds the threshold value, then the processor's color is changed to white and the process time is reduced by 10% (multiplied by 0.9, the new value for the Speedup variable).

Exercise 12-1 Hilltop Steel Works

Background

Hilltop recently opened a new stainless steel rolling plant in Southern Alabama. The new facility includes a milling line that produces large specialty bolts for the offshore and ship-building industries in the area. Demand has exceeded expectations. The line has been operating at a target speed of 600 bolts/hour, but recent orders are calling for rates of 650 to 700/hour. As production has ramped up, the milling machines have become the bottleneck. The bottleneck backs up the forming equipment, which creates serious problems. Two options to help the situation are proposed.

- Increase the size of the buffer area from the forming area to eliminate starting and stopping the forming line.
- Use controls to change the machine speeds dynamically.

It was suggested that a simple simulation would help analyze the problem.

Problem statement

How can the plant respond to market demands?

Operating data

As shown in Figure 12.20, the facility has two milling machines to fabricate the bolts. The rough formed bolts (blanks) come from the forming area to a buffer which sends them on to the milling machine, with the shortest queue having priority. The system was designed for a normal bolt production rate of 600 bolts/hour with the potential to increase that rate. Each machine is designed to run at 360 bolts/hour (10 sec. cycle time). Data has shown that the milling machines have a MTBF of 900 seconds, $\text{exponential}(0,900,1)$, and a MTTR of 200 seconds, $\text{weibull}(200,1,2,0)$.

The conveyors between the blank buffer and the milling machines have a capacity of 20 and a speed of 1ft/sec. Traveling along their x-axis, the bolts take up a space of 12 x 1.5 x 2 inches. The buffer itself has a capacity of 10. Additional conveyors carry the finished bolts away from the machines.

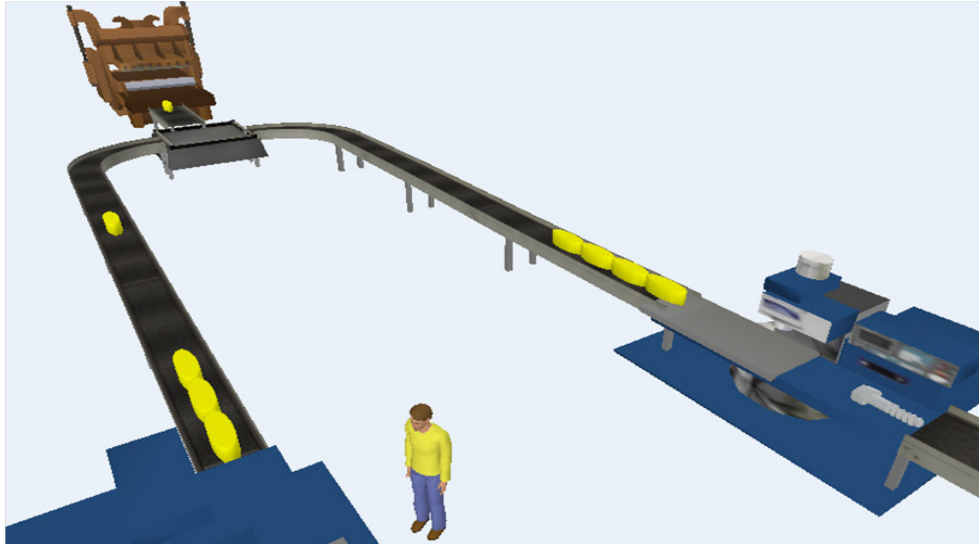


Figure 12.20 Hilltop Steel Works

The machines are new and are driven by servo motors with adjustable speeds. Although designed for a cycle time of 9.5 seconds, the machines may be pushed faster to a cycle time of 7 seconds for short periods of time.

Expected results

Determine the impact of both proposals and recommend a course of action. Check results for the normal production rate and higher values.

Modeling and analysis issues

- Start with constructing a OFD.
- What time unit would be appropriate? Be sure all time-related variables are constants.
- Where should the various speeds (blank production rate, milling machine regular and fast speeds) be stored so that they can be modified easily?
- Before adding breakdowns, how can the simulation be validated to ensure that the correct number of parts are being produced?
- The cycle time of the machine will be dynamic—the processing time trigger is logic based, so where can the appropriate cycle time come from?

- Where can logic be placed to determine if a speed change is necessary? Consider what variable values are needed and what the conditions are to change speeds—both slow to fast and fast to slow.
- Use the experimenter to run replications of 40 hours.

Modeling details are included in the Appendix.

Chapter 12 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Describe some of the important skills of an Advanced User.
3. Describe a hierarchical software architecture and discuss why it is important.
4. Describe the various types of nodes found in the *FlexSim* tree structure.
5. Describe the actions that take place between the time a flowitem enters a processor and when the process time is finished.
6. What programming language is Flexscript most like?
7. What symbol is needed at the end of each statement in Flexscript?
8. Explain why the variable “Carspeed” is not the same as “carspeed”.
9. Write the Boolean logic statement that tests to see if A is equal to B.
10. Write the command for referencing the object connected to the first center port of the current object.
11. Write the command for obtaining the current content of the object connected to the second input port of the current object.
12. Write the command for obtaining the total number of flowitems that have entered the object connected to the third output port of the current object.
13. Write the command to print the value of an integer to the output console.
14. Identify the command that would obtain the numeric value of a label on an object.
15. If a word turns blue when writing in Flexscript, what does it mean?
16. What four steps are needed to create a global variable?

Chapter

13

Communicating Among Objects and Enhancing Model Use

Most simulation languages have a capability to send information from one object to another in order to facilitate building complex logic. They also provide for various means of visually recording variables as the simulation progresses. In *FlexSim*, the communication is accomplished through messages from any one point in the model to another. *FlexSim* also provides a recorder object for monitoring variables. Most importantly, when custom, complex logic is built for objects, those objects can be saved to a user library for future use.

Section 13-1 Communicating between objects

Communicating between objects with messages can be a convenient and efficient way of sending information to coordinate and control the simulation. A message can be sent from any trigger on any object in the model to any other object in the model at any time during the simulation. The OnMessage trigger (located in the Triggers tab) on each object executes when it receives a message.

There are two basic commands for sending messages from anywhere in the simulation:

- `sendmessage(to_object, from_object, param1, param2, param3);`
- `senddelayedmessage(to_object, delay_time, from_object, param1, param2, param3).`

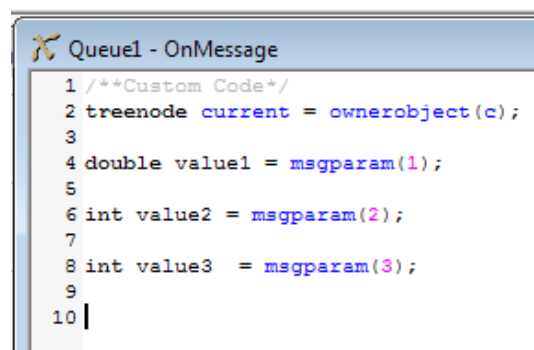


Figure 13.1 OnMessage trigger

Sent messages are received in the OnMessage trigger.

Messages can be sent immediately or after a delay.

Messages are either sent immediately or after a delay. The first command sends a message when the object is executed in the simulation. The delay message command sends the message after a specified delay time.

The `sendmessage()` command sends the message immediately. This means that during the same calculation cycle, the message will be sent and the received message will be executed before the statement following the command is executed. At times such immediate response is required for synchronizing events that are very closely coupled.

There may be times, however, when the received message should be executed only after all other calculations are completed for the time cycle. In such cases, the correct command to use is `senddelayedmessage()` with a delay time of 0. A delay time of 0 will send the message at the end of the current calculation cycle of all objects. It is a good choice when an immediate message is desired but should wait until the states of all objects are updated.

Each message can contain up to three numeric parameters.

On the receiving side, the script in the OnMessage trigger (opened using the puzzle piece icon) contains only a single line of code that defines the treenode variable current. To access the message parameters the command `msgparam(n)`, where *n* is the parameter number 1, 2, or 3 must be added to the logic. This command can be used in an assignment statement, such as `double mp1=msgparam(1);` or directly in the logic, such as `switch(msgparam(3))`.

Messaging examples

Messages expand the scope of logic and allow the simulation of events that would otherwise be difficult. They are often used in simulating events that have to be synchronized. Consider a just-in-time assembly area where products are assembled to order. As a part is being produced to meet a particular order in one manufacturing cell, a subassembly that will be added later has to be started in another cell, as shown in Figure 13.2.

In this case the first assembly unit sends a message at some point in the production cycle to the second unit building the subassembly. A variable containing the order number of the item being built is passed as a message parameter. Using the OnMessage trigger, the second unit looks up what is required and sets up its own parameters to create the required subassembly.

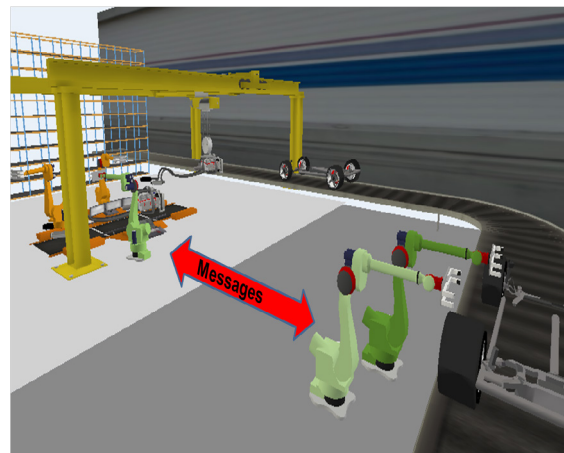


Figure 13.2 Using messages to synchronize operations

Another example of the use of messages involves a machine that, when started from a stopped condition, goes through a speed ramp-up phase. In this case, messages are sent within the same object (the machine) and between objects (a controller and the machine).

The controller, represented in Figure 13.3 as a cube (created using a visual tool), uses its triggers to send start and stop messages to the machine. Whenever the machine receives a start-up message, it begins ramping up its speed. The line graph in the background of the figure (implemented using a Recorder object) shows the machine's speed over time.

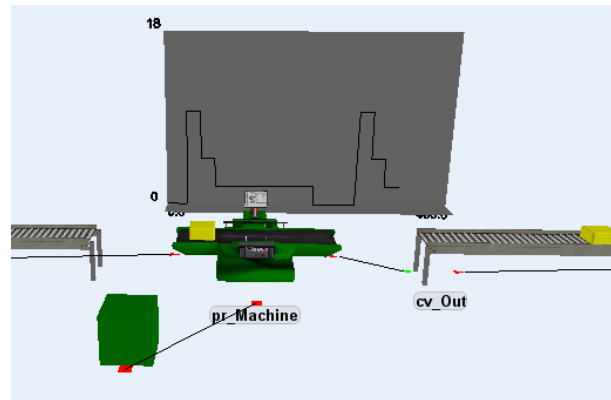


Figure 13.3 Using messages to start and stop a machine as well as ramp up its speed

In this example, as shown in the timeline in Figure 13.4, the controller (a visual tool) is responsible for turning the machine on and off. The machine is scheduled to start 30 minutes after the simulation starts. In order to accomplish this, the controller sends a delayed message to itself through its OnReset trigger. After 30 minutes of simulation, the controller's OnMessage trigger fires and sends a message to the machine to start; it also sends a delayed message to itself to turn the machine off after three hours.

After three hours of simulated time elapses, the controller receives its message and sends a message to the machine to stop; the controller also sends a message to itself to restart the machine in one hour.

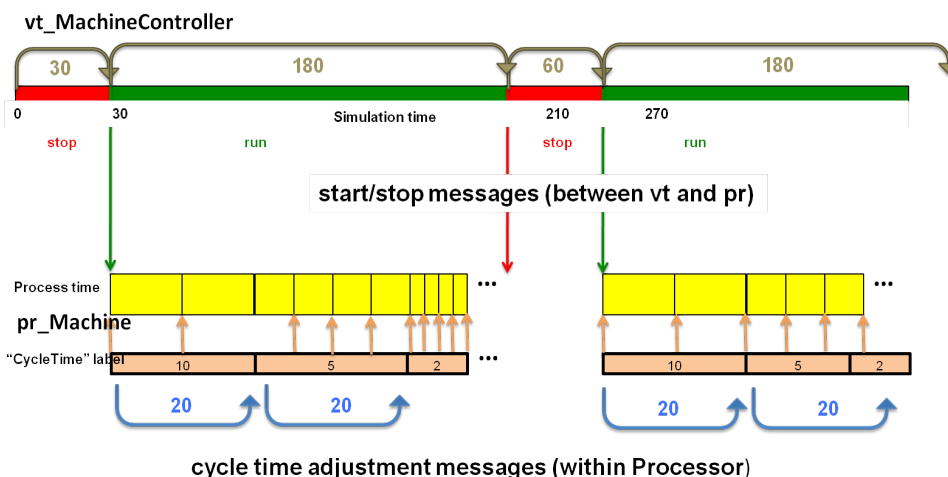


Figure 13.4 Inter- and Intra-object messaging example

Messages can also be used as internal control signals.

As shown in the code segment in Figure 13.5, this process continues throughout the simulation.

```

1 /**Control machine*/
2 treenode current = ownerobject(c);
3
4 switch(msgparam(3))
5 {
6     case 1: // Start machine
7     {
8         colorgreen(current);
9         sendmessage(centerobject(current,1),current,0,0,1); // start message
10        senddelayedmessage(current,180, current, 0,0,2);
11        break;
12    }
13    case 2: // Stop machine
14    {
15        colorred(current);
16        sendmessage(centerobject(current,1),current,0,0,5); // stop message
17        senddelayedmessage(current,60, current, 0,0,3);
18        break;
19    }
20    case 3: // Restart machine
21    {
22        colorgreen(current);
23        sendmessage(centerobject(current,1),current,0,0,1); // start message
24    }
25    break;
26 }

```

Figure 13.5 Controller (visual tool) object's OnMessage trigger logic

Note that in Figure 13.6, the machine (processor object) also uses messages, in this case to represent the ramping up of the machine's speed. The processor sends a series of messages to itself to change the value of its CycleTime label as the ramp-up time progresses. This is accomplished via the case construct in the code segment shown in Figure 13.6.

In this example, the cycle time is 10 minutes for the first 20 minutes after startup, then drops to 5 minutes during the next 20 minutes after startup, and then reaches the normal cycle time of 2 minutes. The machine uses a cycle time of 2 minutes until the next stoppage and restart.

The third message parameter is used to track the startup phase. Note that the starting and stopping of the machine is controlled by opening and closing, respectively, the processor's input ports. The `stopobject()` and `resumeobject()` commands could not be used, because if an object is stopped, it cannot, among other things, send and receive messages.

The effect of this ramp-up process is illustrated in Figure 13.7 using a Gantt chart representation of the "life" of flowitems in the model. For each flowitem, the leftmost or blue horizontal bar is the time the item spends on the input conveyor;

```

1 /**Run start up ladder*/
2 treenode current = ownerobject(c);
3
4 /**Startup Ladder*/
5 int stepnum = msgparam(3);
6
7 switch(stepnum)
8 {
9     case 1: // start machine
10     {
11         openinput(current);
12         colorgreen(current);
13         setlabelnum(current,"CycleTime", 10);
14         senddelayedmessage(current,20, current,0,0,2);
15         break;
16     }
17     case 2: // next step
18     {
19         setlabelnum(current,"CycleTime", 5);
20         senddelayedmessage(current,20, current,0,0,3);
21         break;
22     }
23     case 3: // final step
24     {
25         setlabelnum(current,"CycleTime", 2);
26         break;
27     }
28     case 5: // stop machine
29     {
30         closeinput(current);
31         setlabelnum(current,"CycleTime",0);
32         colorred(current);
33         break;
34     }
35     default:
36         msg("Error", "Invalid Case value at Machine");
37         stop();
38 }
39

```

Figure 13.6 Machine (processor) object's OnMessage trigger logic

the center or green bar is the processing time of the machine; and the rightmost or red bar is the time the item spends on the output conveyor. Notice the intended decrease in processing time over time due to ramp-up; also, the time on the input conveyor decreases due to the machine becoming available and the processing time speeding up.

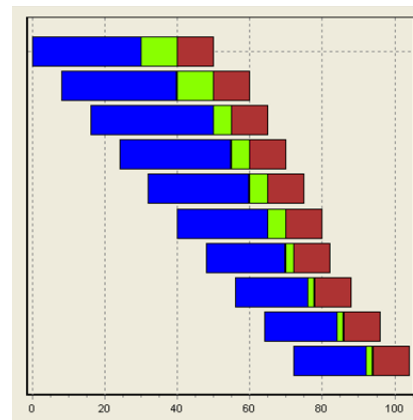


Figure 13.7 Effect of ramp-up on Machine's processing time

A label on an object can also be configured to hold table data.

Section 13-2 Label Tables

The Intermediate User utilized labels to hold individual pieces of information; however, labels can also be used to hold arrays of data. Such a label is called a label-table and is useful in the more advanced methods of managing where data can be stored in the simulation.

A label-table is created using the Label tab on the object and selecting to add either a Number Label or a Text Label. Then select the LabelTable button, and fill the cells with data.

Addressing the label-table is similar to addressing a global table using commands such as `gettablenum()` and `settablenum()`; however, unlike global tables, which can be identified by name, the label-table has to be identified by its label name and the command

```
label(object, "labelname").
```

To obtain a value from a label-table that exists on the currently selected object from one of its own triggers, use the command

```
gettablenum(label(current, "labelname"), row, column);
```

If the label-table is on a separate item, then item would be used in the place of `current`.

To get the value of a label-table that is on an item that is the first or last item in a queue or a combiner, use the command

```
gettablenum(label(first(current), "labelname"), row, column);
```

The label-table is useful when related data is required dynamically. An example would be a distributor that makes up boxes based on a real time order entry system. The number of items in a box can vary for each order. In the simulation shown in Figure 13.8, bottles from four entry lines are combined to meet custom orders.

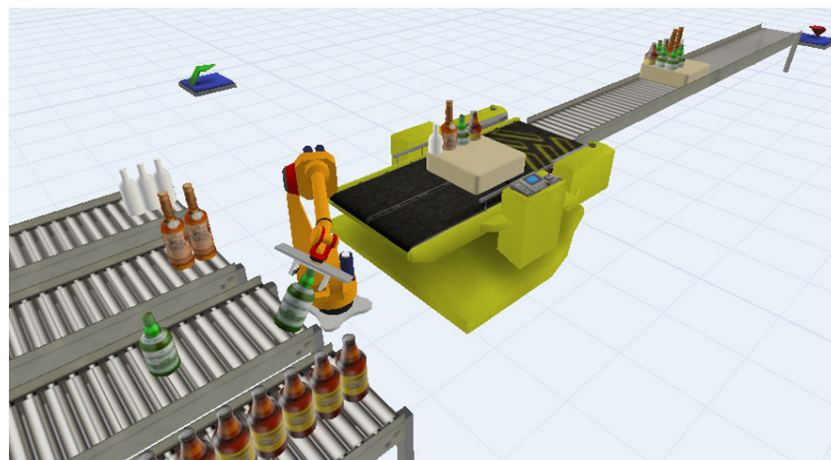


Figure 13.8 Label-table example

A numeric label-table called “Wines” is created on the flowitem that represents the box being filled. The table has 4 rows and 1 column. The box source, connecting to the combiner, fills out the label-table on the flow item with the number of bottles needed from each entry line to represent the current order.

In the OnEntry trigger of the combiner, the standard code for updating the combiner’s component list is modified to use the label-table instead of a global table as was done in previous chapter exercises. The resulting code is shown in Figure 13.9.

```

treenode thelist = getvarnode(current, "componentlist");
treenode thesum = getvarnode(current, "targetcomponentsum");
setnodenum(thesum, 0);

for(int index=1; index<=nrows(thelist); index++)
{
    setnodenum(cellrowcolumn(thelist, index, 1), gettablenum(label(item, "Wines"), index, 1));
    inc(thesum, gettablenum(label(item, "Wines"), index, 1));
}

```

Figure 13.9 Label-table coding example

Section 13-3 The Recorder

The recorder object provides a visual record of a variable’s value as a model is executed. It was used in the example earlier in the chapter to track the machine’s speed while it was ramping up. In that case it was referencing a label on the machine object.

When the recorder object is first brought onto the modeling surface and opened, it requests the type of data that is to be recorded. The types of data include the following:

- *Table Data:* Plots data already in a global table that is populated during the execution of the simulation
- *Standard Data:* Plots data that is already automatically collected on an object and is shown under the object’s Statistics tab
- *User-Defined Data:* Allows the graphing of specific information collected during the run of a simulation. There are several options for the type of output to display (line graph, histogram, etc.). The type of display should be consistent with the type of data being collected. The data that will be recorded is identified using the Browse button and choosing a node from the tree view.

Details for setting up a line graph used in the ramp up example are given in the Appendix.

Recorders can display data in real time as the simulation is running.

Section 13-4 Customizing object placements

The default graphics for objects in *FlexSim* are normally sufficient for visualization; however, there are times when changing the placement of flowitems and the visual representation of the object itself would benefit the simulation. For example, the processor by default moves flowitems across the top of the 3D object; however, if the processor represents a bank teller behind a counter and the flowitem is a person being served, then the default visualization would not be representative of the system and may cause confusion for a layperson.

Changing the visualization may involve both the object and the flowitem. The following example converts a processor's graphic representation from a machine to a service desk.

As shown in Figure 13.10, change the graphic as follows:

- Open the processor and uncheck “Convey Items Across Processor Length”
- Change the processor to some other shape on the General tab—for example a box. A the box is in the FlexSim5/fs3d/general folder. You may have to navigate to the folder or select some other 3D shape from an external source such as Google 3d Warehouse (.skp).
- Modify the size on the General tab as desired.

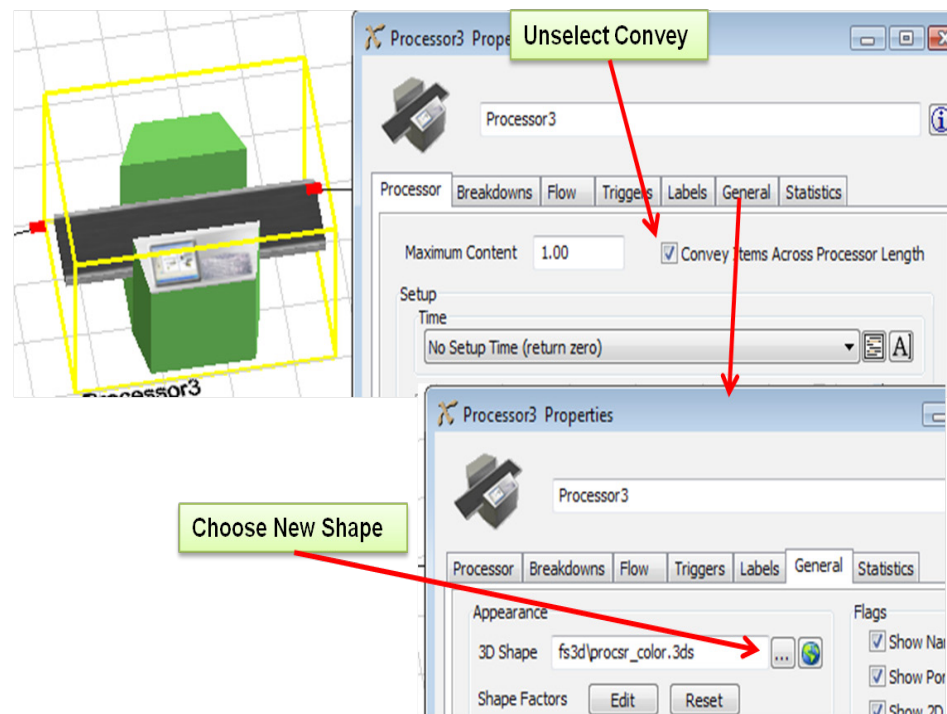


Figure 13.10 Changing a processor's shape

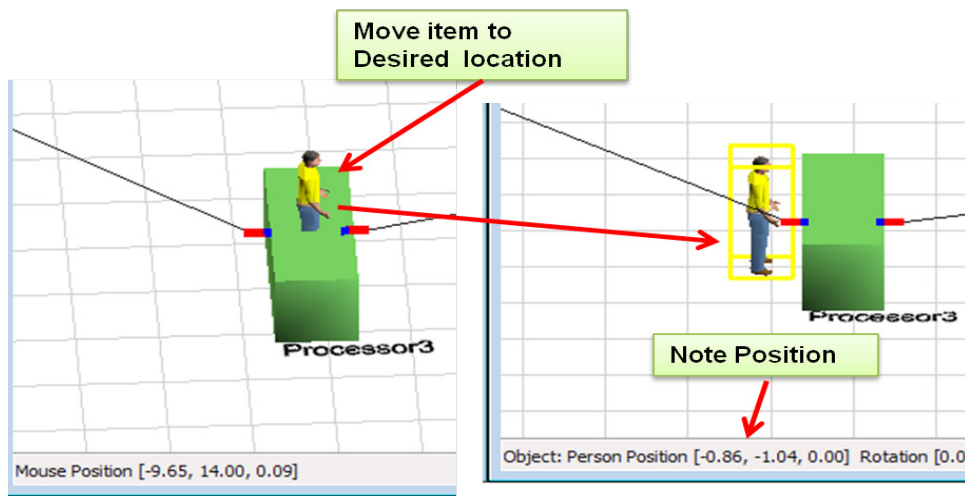


Figure 13.11 Changing the flowitem's position at the processor

To set the position of the flowitem, as illustrated in Figure 13.11, do the following steps:

1. Use a source to supply the processor and select a person as the flowitem.
2. Start the simulation and stop when the person is in the processor shape.
3. Move the person with the mouse, or use the General tab on the flowitem.
4. When the position is correct, note the location at the bottom of the screen when the flowitem is highlighted.
5. Use the position values in a `setloc()` command in the OnEntry trigger of the processor.

In a similar way, a task executer, such as an operator, can be shown to push a flowitem rather than carry it. The steps are outlined below and illustrated in Figure 13.12:

1. Run the simulation until the operator is carrying the flowitem.
2. Move the flowitem with the mouse, or use the General tab on the flowitem.
3. When the position is correct, note the location at the bottom of the screen when the flowitem is highlighted.
4. Use the position values in a `setloc()` command in the load trigger of the operator.

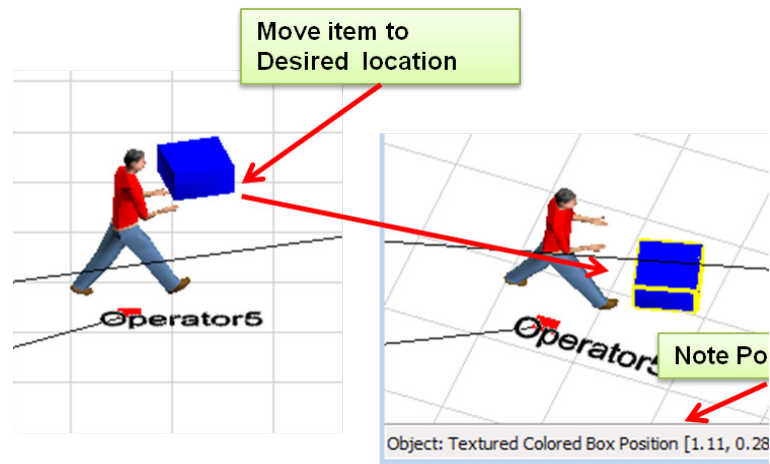


Figure 13.12 Setting the flowitem relative to the task executer

Section 13-5 Reusing custom objects

Objects can be saved in custom libraries for future use.

The Advanced User will often create objects with new images or shapes attached to them or include custom logic on the object, such as the machine seen earlier that had its own built in startup cycle. Such objects may be desired in other simulations, and significant time can be saved if custom work can be reused. In *FlexSim*, user libraries can be created and used to share objects.

Clicking on the New Library icon at the top of the Library tool bar opens a blank library. An object can be added to the library by first right clicking on the object and selecting Edit. At the bottom of the secondary window that opens is an option to add the object to the library. The procedure is shown in Figure 13.13. Note that objects can't be added to the standard *FlexSim* discrete and fluid libraries. The user library can be saved and re-opened at any time. Objects in a user library are brought onto the simulation surface by clicking and dragging.

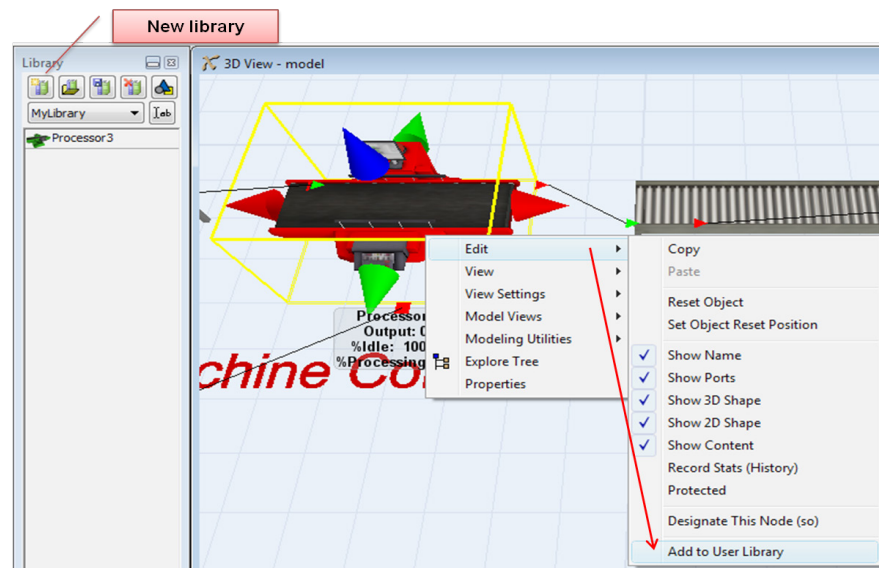


Figure 13.13 Saving an object to a user library

There may be a need to save a group of objects such as a manufacturing cell similar to the one shown in the label-table example in Figure 13.8. Objects can be maintained together and attached to a visual tool object. The steps, as shown in Figure 13.14, are as follows:

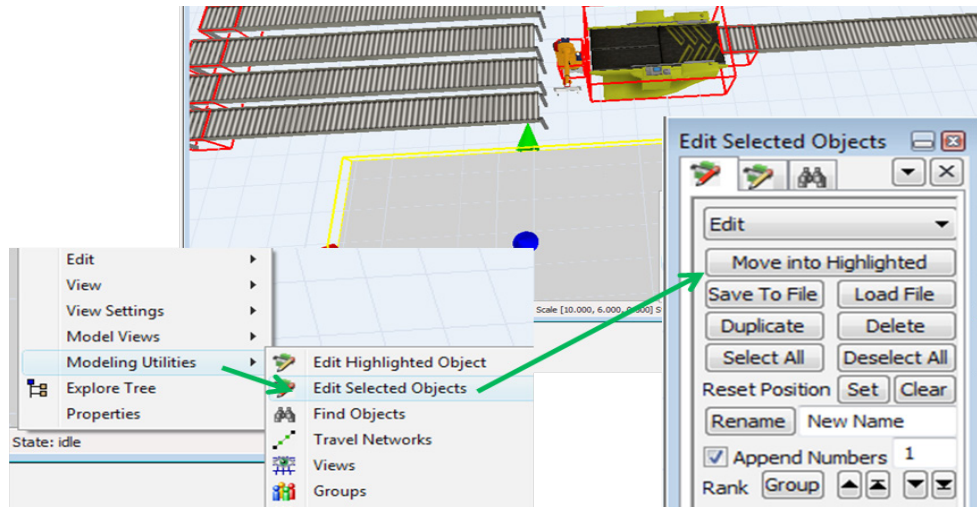


Figure 13.14 Grouping objects in a visual tool

1. Select the objects to be grouped by shift-dragging the mouse over them. Selected objects are enclosed by a red cube in FlexSim.
2. Highlight the visual tool by clicking on it.
3. Right click on the model field and select Modeling Utilities followed by Edit Selected Objects.
4. In the Edit Selected Objects window, select Move into Highlighted.

The objects will all move as a group with the visual tool. In addition, the visual tool can be saved to a user library for reuse in other simulations. When the visual tool is brought onto the simulation surface, all of the objects on the visual tool will also appear. To ungroup the object from the visual tool, select the objects but do not make any selections before clicking on Move into Highlighted.

Exercise 13-1 Fister's Express

Background

Fister's Frozen Foods, introduced in Chapter 4, wants to offer quick-delivery frozen food combinations that will provide a week's worth of ready-to-prepare meals. Customers will be able to choose a combination of meats, fish, poultry, vegetables, and deserts. Fister's guarantees that the order will be filled and shipped the same day.

Objects can be grouped for editing and moving.

To provide the most efficient service, the orders will be assembled automatically and moved to the existing shipping area. Control of the automated system is being left to an engineering contractor.

Operating Data

Marketing expects orders initially to arrive at an average rate of 5 orders per hour (exponentially distributed) but that the order rate

could increase to one order every 9 minutes, on the average. Customers can pick a combination that contains choices from five categories. The maximum number of choices (there is no minimum) in any given category is displayed in the table:

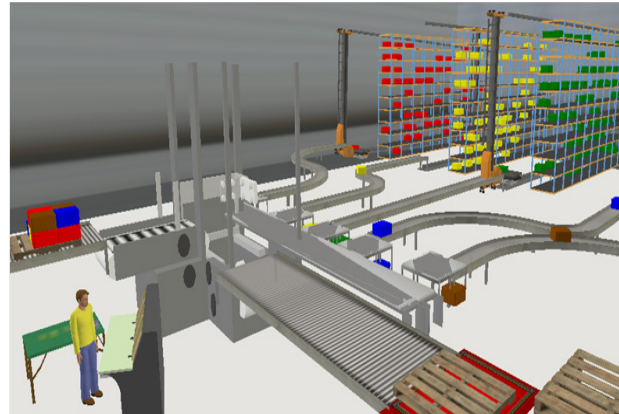


Figure 13.15 Fister's automated order assembly area

Category	Maximum number of choices
Meat	5
Chicken	3
Fish	3
Veggies	4
Desserts	2

Each category is stored in a separate aisle in the frozen food warehouse. Each aisle has its own robotic picker. The pickers require a base travel time to reach the first item and then an additional amount of time to collect the other items in the order.

Total travel times are:

Category	Base Time	Additional Mean	Additional Std. Dev.
Meat	0.5	0.6	0.3
Chicken	0.7	0.4	0.2
Fish	0.85	0.8	0.4
Veggies	0.6	0.5	0.3
Desserts	0.5	0.3	0.15

Note: All times are in minutes. The base time includes travel to and from the area of the first item. The additional time is normally distributed.

Orders are maintained in a queue as they are received. When an order is ready to be processed, a pallet is placed at a holding station with an order assigned to it. Picking orders are sent to the appropriated robots. Once all the picking stations have

completed their task the pallet is released to travel to the assembly station where the order is placed on the pallet. The assembly task requires 15 seconds (exponentially distributed). The full pallet then travels to the shipping room.

Conveyor travel times are:

Conveyor	Travel time (minutes)
Steak pick station to assembly	1
Chicken pick station to assembly	0.833
Fish pick station to assembly	0.8
Veggies pick station to assembly	1
Desert pick station to assembly	0.8
Pallet holding to assembly	2
Assembly to shipping	2

Expected results

Start with an OFD (object flow diagram), build the model, and run the simulation for 7 days (24 hour operation) and comment on such performance metrics as:

- How long does it take to complete an order?
- How many orders are completed?
- How many orders are backlogged?
- What order rate can be reasonably achieved?

Modeling and analysis issues

- What level of detail is needed? What time unit is used?
- Where will the orders be placed as they come in? What can represent an order?
- How will the conveyor travel times be implemented?
- How should the data be managed? What tables are needed? Could a label-table be used?
- What information should be sent to each picking robot? Where should it be sent from?
- How will a void in a particular category be handled?
- How will the robots report back when they complete their task?
- What are the initial conditions for each piece of equipment?
- How will the order time in the system be calculated?
- How can the order correctness be checked?
- What visualizations can be used?

Exercise 13-2 Peoples Surgery Center

Background

Given the changes in medical care being directed by the government, Carson Memorial Hospital has decided to open a surgery center for common out-patient surgical procedures. It is felt that such a center could be operated more efficiently and cost effectively than what is currently offered. The hospital was originally designed for major procedures and longer-term care.



Figure 13.16 Peoples Surgery Center – reception and surgery waiting rooms

Before setting up the new facility, the hospital administration wants to have a simulation developed to check on the staffing levels that are needed and the amount of time patients would spend in the facility—especially in the admission and lab areas.

Problem statement

Provide a simulation to help establish operating conditions for the admissions and lab areas of the surgery center.

Operating data

Three types of patients are anticipated to use the new facility:

1. Those who arrive for the surgical appointment with all the required laboratory work completed.
2. Those who have appointments but need laboratory work before beginning the surgery.
3. Those who are using the lab facility for routine tests or pre-screening tests for a later surgery.

Patients who arrive at the facility enter a waiting room and are called by an admissions agent. Patients who are ready for surgery (Type 1) are taken first followed by Type 2 and Type 3 patients.

After admissions, patients going for their surgery continue to a small waiting area to wait to be taken to the surgical rooms. From there they must be accompanied by an orderly to the surgery area.



Figure 13.17 Peoples Surgery Center – lab area

Patients who need lab work can walk to the lab area after admissions. Once at the lab, they

- Register at the lab admitting area (one lab agent).
- Wait in the lab waiting room until they are called by a lab technician.
- Patients only there for lab work can then go home when lab procedures are complete.
- Patients continuing for surgery (Type 2) return to the main waiting area where they are called for surgery admittance.

Admissions operating hours

- Opens at 7 a.m. for patients with surgery appointments (Type 1 and 2).
- Patients for lab work arrive between 10 a.m. and 3 p.m.
- No new patients are allowed to register after 3 pm.
- The admissions/lab areas are available until all patients who entered from 7 a.m. to 3 p.m. are served. It is planned that these areas will be clear of patients by 7 p.m..

Patient Type	Percent before 10	Percent after 10	Admission mean time	Distribution
1	85	40	12	Normal, S.D. = 1.5
2 before lab	15	10	40	Normal, S.D = 4
2 after lab work			3	Normal, S.D = 1
3	0	50	30	Normal, S.D. = 3.2

On normal days the mean time between arrivals for admitting is 12 minutes, distributed exponentially. However, the time between arrivals could decrease to 10 or even 8 minutes. Admitting rates, the percent of each patient type, and process times are provided below, along with travel times and lab processing times are. All times are in minutes.

Travel times	
Routes	Distribution
Admissions to Surgery	Uniform (min = 3, max = 8)
Between Admissions and Lab	Uniform (min = 2, max = 5)
Surgery to Admissions (orderly)	Constant = 3

Lab times	
Procedure	Distribution
Lab admission	Triangular: min = 2, max = 5, mode = 3.3
Lab sample	Triangular: min = 6, max = 12, mode = 5.1

Current staffing levels are

2... Admissions clerk

2 ...Orderlies

1... Lab clerk

1 ...Lab technician

Expected results

- Start with an OFD
- Run the simulation for 20 days and comment on such performance measures as
 - The average wait time for each waiting area.
 - How many patients per day can the facility accommodate? Will patients still be in the area after 7 p.m.?
 - How long are patients in the simulation?
 - What is the impact of different staff levels or other changes?

Modeling and analysis issues

- How can an entire system be modeled in sections to test it before putting it all together?

- Getting through registration
 - What are different ways to bring the patients into the system? How can you regulate the times the center allows patients to enter?
 - How can you identify the patient type and flag the type 2 patients who have come back from their lab work?
 - What queue options will prioritize how patients leave the main waiting area?
 - What logic is needed to send the patients to the right place?
 - How can you keep track of each type of patient and the total number of patients?
- Moving patients to surgery
 - How can the orderlies be shared?
 - How can the travel times be set up to be different in each direction?
 - When an orderly takes a patient to the surgery area, consider how to make sure the orderly returns to the surgery waiting area.
- Lab area
 - How will patients move to the lab at the appropriate speed?
 - Patients coming into the lab waiting room are either waiting for a receptionist or a lab technician. How do you identify them and make sure they go to the right place?
 - How will only the patients going for a lab sample have a technician take them?
- What variable on a flowitem can be used to track a patient's time in the system?
- In terms of visualization, how can the orderly or technician walk side by side with the patient?
- What are the startup and finish times for the simulation runs?
- How can you tell if everyone is finished at the end of the day?
- How would you add staffing?

Additional modeling details are included in the Appendix.

Chapter 13 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Discuss the importance of communication between objects.
3. Describe two uses for communicating among objects in a simulation.
4. Describe the difference between sending a message immediately and sending it after a delay. Provide an example where each might be used.
5. Describe two examples where you might send a message from an object to itself.
6. Describe the importance of label-tables. Provide an example where one might be used.
7. Describe an example where a recorder object might be used.
8. If you change the graphical image of a processor and want to place the entering flowitem in the proper location, write the command and indicate where it should be placed.
9. Discuss how the user library can be used in simulation projects.

Chapter

14

Simulating Fluid Flow

The majority of simulation application packages on the market are based on discrete-event simulation; however, interest in simulation has grown to include operations involving the continuous flow of material such as liquids, powders, and high speed bottling. Simulating both a processing and packaging step in a production line involves the transition between continuous and discrete events. A few simulation packages only simulate fluid flow, just as some only simulate discrete events. Others, like *FlexSim*, are capable of simulating both fluid and discrete systems in the same simulation model.

Section 14-1 Basics of fluid-flow simulation

Fluid Concepts

A material flow is considered to be a fluid when its moving material cannot be identified by individual objects but appears as a continuous body. Fluid, however, doesn't necessarily mean liquids. Liquids, gases, and small particles such as rice, cereal, and sand are obvious examples of fluid flow. A less obvious example of a fluid is a high-speed bottling line (> 1600 bottles per minute), it can be thought of as either a fluid or a discrete material flow. A fluid (or continuous) flow is usually described in terms of units per time, such as gallons per minute.

Being able to simulate fluid flow is important to industries that have both continuous and discrete manufacturing processes. The manufacturing involves a process area where fluid-type materials are produced as well as a discrete area such as a packing line. Discrete material flow has specific events associated with it, such as entry into a machine or exit from a queue. These events are placed into an events

list for processing in time order by the discrete-event simulator. Fluid flow, however, moves continuously.

To keep track of the continuous nature of fluid flow in a discrete-event simulator, a clock (called a *ticker* in *FlexSim*) creates events at set intervals of time, as illustrated in Figure 14.1. The intervals need to be small enough to capture all of the characteristics of the fluid flow. At each ticker event, the condition of each fluid object is updated by calculating the amount of fluid moved from one object to the next. As shown in Figure 14.2, the calculation has to take into account the material available, the space available, and the input and output rates. A recursive calculation is continued along the complete fluid network.

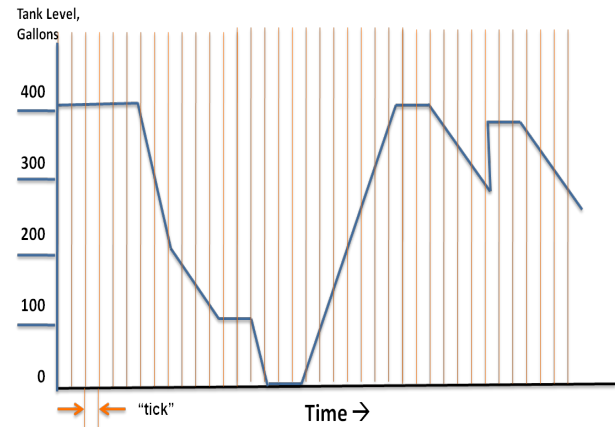


Figure 14.1 Fluid flow state by tick event

In *FlexSim*, when the first fluid object is brought onto the simulation surface, the ticker is automatically added. The ticker can be moved to any position on the screen.

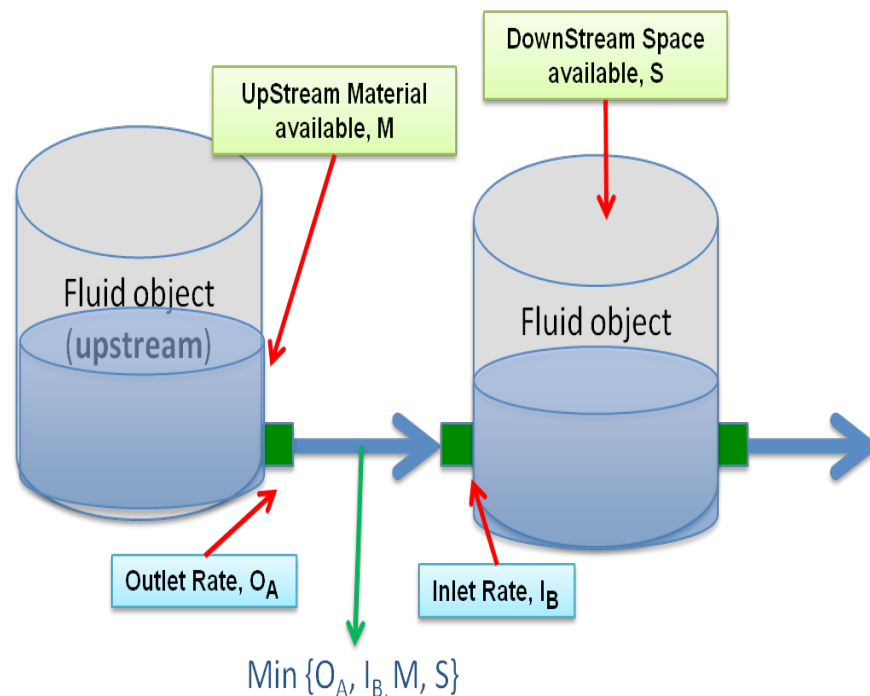
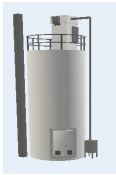


Figure 14.2 Calculating fluid flow

Section 14-2 Fluid objects

In *FlexSim*, the fluid objects are found in a separate library in the library icon grid and are located by using the library drop-down menu. Fluid objects are similar in basic function to their discrete counterparts. The difference comes in setting their specific variables.

Fluid Generator



Discrete Analogy: Source

Function: Creates fluid material

Operation: Provides an infinite supply of fluid material. Tank will either continuously refill at a fixed rate or run empty and then refill after a specified delay.

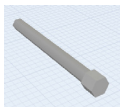
Fluid Terminator



Discrete Analogy: Sink

Function: Removes material

Fluid Pipe

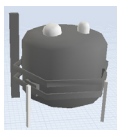


Discrete Analogy: Conveyor

Function: Move material from one object to another

Operation: Provides a transfer object for fluid. Fluid starts flowing out of the pipe once it becomes full. If the pipe empties, fluid will not flow until the pipe fills again. The output rate will try to match the input rate (up to the maximum pipe rate). The output pipe has three options to direct flow: First Available sends the most fluid possible to the first available downstream object; Flow Evenly tries to balance the flow to all downstream objects; and User Defined allows the user to designate the flow split.

Fluid Tank

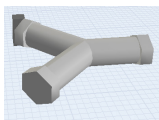


Discrete Analogy: Queue

Function: Stores fluid material

Operation: Acts as a surge for fluid material. Triggers are available when tank level reaches specified levels.

Fluid Blender

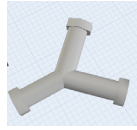


Discrete Analogy: Combiner

Function: Blends fluids into a single material based on a fixed percentage

Operation: Used for in-line, continuous blending of fluid streams. The blender will automatically adjust input rates to maintain specified mix percentages.

Fluid Splitter

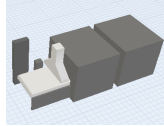


Discrete Analogy: Separator

Function: Splits flow based on fixed percentages

Operations: Used for in-line, continuous splitting of a fluid to different downstream objects. The splitter will automatically adjust output rates to maintain the specified split percentages.

Fluid Processor

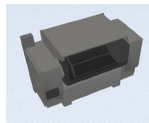


Discrete Analogy: Processor

Function: Processes fluid

Operation: Simulates the continuous processing of a fluid. Once empty, the processor will only start flowing out when it is full. Fluid loss in the processor can be specified as a percent of the input rate. The processor has only one input and one output.

Fluid Mixer

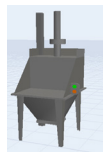


Discrete Analogy: Multiprocessor

Function: Combine materials in a batching operation

Operation: Maintains a processing Step Table that includes actions taken at each step, any delay times, and a Recipe Table that includes the source, amount, and addition step for each material used. Material leaves the mixer only when the batch is complete.

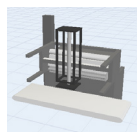
Item to Fluid



Function: Converts a flowitem to a fluid

Operation: Simulates emptying individual containers of fluid materials by converting the material content of each flowitem into a fluid stream.

Fluid to Item



Function: Converts an amount of fluid to a flowitem

Operation: Simulates packing a specified amount of fluid material into a container.

Section 14-3 Specifying Fluid Operations

Fluid objects have more robust choices than their analogous discrete objects. The fluid tank is typical of the many flow choices available. Some of the options available on the Tank tab are shown in Figure 14.3 and described below.

Both the maximum and initial content can be specified. Note that the input and output ports have both a total maximum object flow rate as well as a port maximum. These limits regulate the allowed flow out of the object. It is possible that the sum of the maximum port rates be set to total more than the object rate. However, the default flow logic out of an object gives priority to the first port, then the second, and so forth up to the object maximum flow rate. Other flow logic options can be selected using the drop down choices in the Adjust Output Rates drop down menu. The port scale factors appear when inputs and outputs are connected. The factor sets each port rate to a fraction of the maximum port rate.

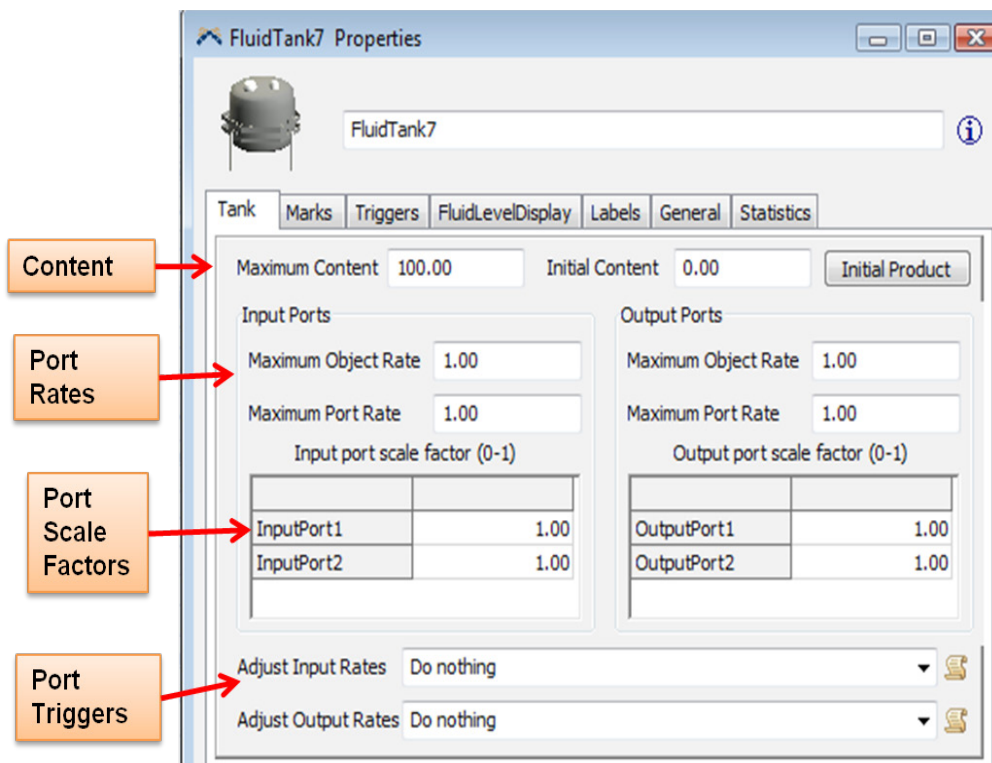


Figure 14.3 Fluid tank display

Special triggers are available on this interface in addition to those located on the Trigger tab. The triggers here are executed at each clock tick. One special function available from the drop down menus is the Open One Input (or Output) choice. For inputs, the object will choose one upstream object at a time and use it until the object is empty. For outputs it will flow to one object at a time until that object is full.

Fluid ports, unlike discrete object ports, can be managed independently. For example, individual fluid ports can be opened and closed. Consequently additional commands are available:

- `openallip()`, `closeallip()`, `openallop()`, and `closeallop()` are better choices than the discrete equivalent `openinput()` for opening or closing all input or output ports at the same time.
- `openip()`, `closeip()`, `openop()`, `closeop()` commands can control individual ports (they will effectively change the port scale factor to a 1 or 0).
- Examples: `openallip(object)`; `openip(object, n)` where `n` is the port number

As shown in Figure 14.4, each fluid object has a level indicator. The indicator's attributes are found on the FluidLevelDisplay tab. Checking the box to Draw level indicator will result in the indicator being visible. The shape and position of the indicator can be changed by using the shape factors.

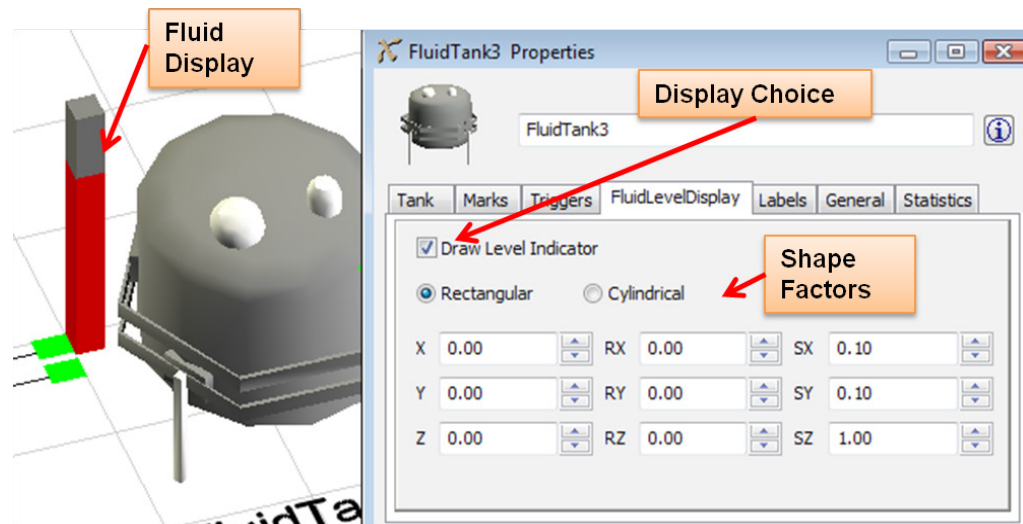


Figure 14.4 Fluid tank level display

With all the variables associated with fluid objects, there are a few points to keep in mind:

- Pipes, like conveyors, are not required to move material from one object to another.
- When there are multiple fluid objects in series, be sure that the rate settings are correct in order to avoid creating an unintended restriction to the overall flow.
- The capacity of an object *must always be greater* than the amount of material that will flow in one tick. This is seldom a problem if the time unit is seconds, but may be a factor if the time unit is one minute.

Mixer object

The fluid mixer is probably the most versatile fluid object. It represents one of the most important of all fluid operations - batch processing. The use of both a step and recipe table to describe a batch operation is typical of process industry practice for describing the mixing process.

While the Mixer tab contains the normal fluid object variables, the Steps tab provides the batch description. The Steps tab, as shown in Figure 14.5, has three separate sections: Mixer Steps, Mixer Recipe, and mixer triggers. The Mixer Steps section allows you to add or subtract steps by using the Number of Steps field, and update changes by clicking the Update button. Changes will be reflected in the table below. The Description is a text field for descriptive use only. The Delay time, in simulation units, occurs either after any addition listed for that step or immediately if no addition is listed for the step.

The Recipe table contains the ingredients along with the port through which they enter the mixer. Each material entry has a step number associated with it to indicate when in the steps the material should be added. The ingredient field is a text field for descriptive use only. The number of recipe entries can be changed by adjusting

the value in the field and clicking the Update button. An entry is made for each time a component is added to the mix. Note that in the example water is added more than once in the recipe and that the recipe list doesn't have to be organized by step number.

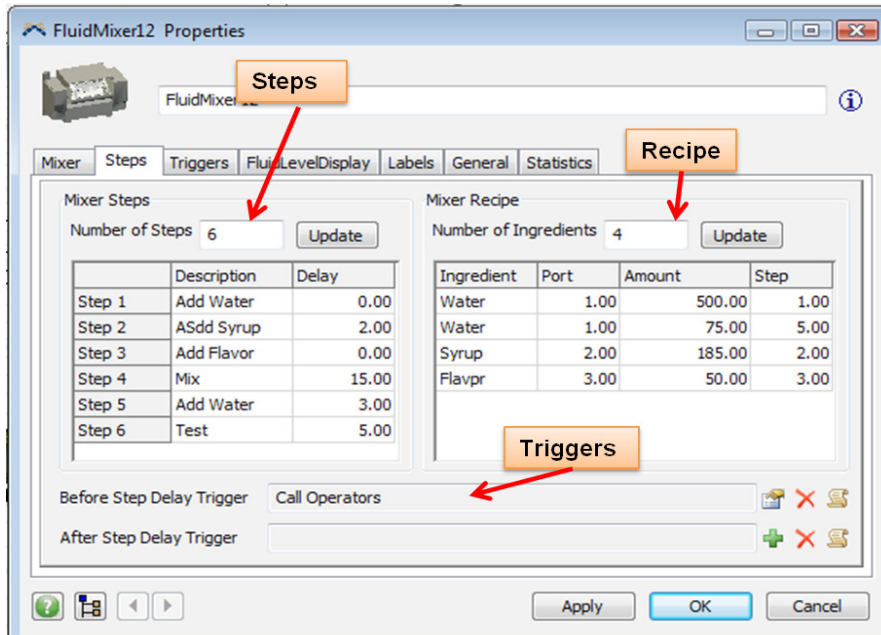


Figure 14.5 Fluid mixer description

The recipe tab also contains triggers that execute before and after each step. The figure for the Steps tab shows a sampling delay in step 6 and an operator being called. The operator is called using a picklist choice for the trigger Before Step Delay. When the material addition in step 6 is complete, the operator is released from duty by using the picklist choice for the After Step Delay Trigger. Other custom logic may also be used in these triggers.

Exercise 14-1 James Peanuts

Background

James Peanuts, a family owned and run business, has seen significant growth over the last few years. After producing a homemade gourmet potato chip for many years, James finally gave in to his advisors, who recommended expanding their line with “flanker” products—namely flavored peanuts.

The small manufacturing group came up with a design for producing the new products that they felt was flexible and lean. It involved a batch makeup system of flavoring to add to the peanuts. Regular peanuts could be run on



Figure 14.6 Nuts by James

the line while the flavor system was cleaned for the next product. Everything went well until production started increasing and the flavor system became a bottleneck. The group could not understand the problem since their calculations showed the batch make-up time was short enough to handle the higher rates. As a member of the manufacturing team, you decided to use simulation and take a different approach to the problem.

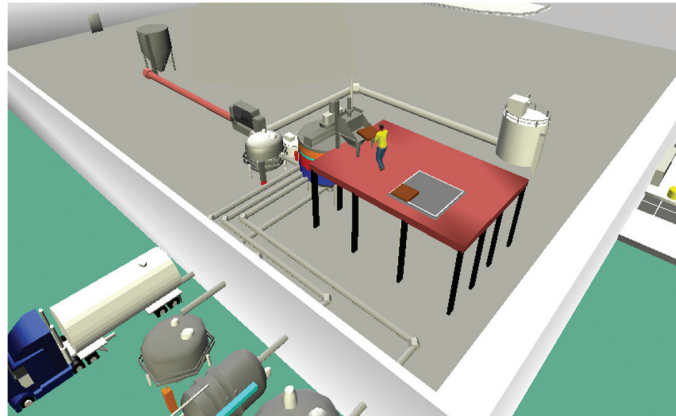


Figure 14.7 Flavored peanut production

Problem statement

What will it take for the flavor system to run at higher production rates without becoming bottlenecked?

Operating data

Flavor additives are made up in a mixer that gets material from three liquid flavor tanks. Additionally, a powder additive is dumped into a hopper by an operator and metered into the mixer. The mixer mixes 4,325 pound batches that are transferred to a smaller use tank that acts as a buffer to maintain a continuous flow into a machine that coats the flavor onto the peanuts.

The plant follows the general practice of using pounds as a common measurement, thus eliminating density conversion factors when dealing with liquids. Also, mass flow and weight measurements are more common and accurate than volumetric meters.

Flavored peanuts are produced by feeding the peanuts, after they're roasted and taken from the shells, into a blender (capacity 20 pounds) where the flavor coatings are sprayed onto them. The weight percentage of the flavored product is

- 60% peanuts
- 40% flavor

Normal peanut production is 95 lbs/min (1.583 lbs/sec).

Flavoring additives 1, 2, and 3 are maintained in storage tanks that are automatically refilled. The powdered flavoring comes in 40 lb sacks that are dumped by an operator into a hopper next to the mixer. Details of the tanks are:

Unit	Capacity (lbs)	Initial Level	Discharge Rate (lbs/min)	Discharge Rate (lbs/sec)
Flavor 1	1500	1500	128	2.133
Flavor 2	2000	2000	130	2.167
Flavor 3	1000	1000	105	1.750
Dry mix hopper	400	0	143	2.383
Use tank	3300	3000	150 max	2.500

The recipe follows standard nomenclature and is divided into tables of steps and components; the tables are provided below. Ingredients are added individually. Once the mix cycle is complete, the material can transfer to the use tank. The transfer pump is either on or off. The mixer cycle is automatic and starts again as soon as a batch is transferred. The mixer transfer rate is 570 lbs/min (9.5 lbs/sec).

Ingredient	Recipe Table Amount (lbs)	Step
Flavor 1	1460	1
Flavor 2	1605	2
Flavor 3	932	3
DryMix	328	4

Step	Step Table Description	Delay (sec)
1	Add Flavor 1	0
2	Add Flavor 2	0
3	Add Flavor 3	0
4	Add DryMix	0
5	Mix	420

The plant's lean team calculated that they could make and transfer a batch approximately every 45 minutes, which could support a dry peanut rate of over 130 lbs/min. Since the normal rate was 95 lbs/min of dry peanuts, lean team decided that they would reduce the buffer (use tank) size to 3300 lbs. It is important that the use tank not run dry so as to stop the peanut flow as it reduces production and may degrade the peanuts.

To save wear and tear on the mixer's transfer pump that would be caused by excessive cycling, the maintenance group set up level triggers on the use tank. When the level gets to 2000 lbs, the transfer pump turns on to fill the tank from the mixer. The pump is turned off when the level in the tank reaches 3200 lbs. The maintenance group want at least 500 pounds between the high and low levels.

Expected results

Starting with the standard dry peanut rate, determine when the flavor additive system becomes bottlenecked. Consider rates from the base of 95 to 120 lbs/min of dry peanuts. Run the simulation for 2 shifts (16 hours), the typical run cycle. Start with an OFD of the system. Explain why the estimates of the lean team were incorrect.

Modeling and analysis issues:

- What assumptions can be made about the operation?
- What time unit should be used and why?
- Should the peanut flow be fluid or discrete?
- Are pipes needed to connect equipment?
- How are bags of dry flavoring transferred to the mixer?
- What performance measures should be calculated?
- Are there improvements that can be made that do not require any physical changes to the equipment?
- What visualization elements can be used?
- How can the dry peanut rate and other variables be easily changed from one location?

Exercise 14-2 Western Grain

Background

The Western Grain Cooperative is planning a new grain loading facility to transport grain from their main silos to barges on the Mississippi River. Grain is loaded onto barges at the facility's four docks. Before they go ahead with the plans, the members of the board want to be assured that the concept will work and that grain will not be left in the main facility. They want to be sure that they can ship between 55,000 and 60,000 tons of grain per day.



Figure 14.8 Western Grain Co-op

Problem statement

Determine the feasibility of the loading operation including the number of tugs that are required.

Operating data

Empty barges are dropped off at a holding area where they are picked up by tugs and transported to the docks. When the barge is full, it is picked up by a tug and moved to a holding area to await transportation down river.

Grain is discharged from the storage area at a fixed rate of 40 tons per hour, and distributed to one barge at a time until the barge is full. Each dock can load at the same rate.

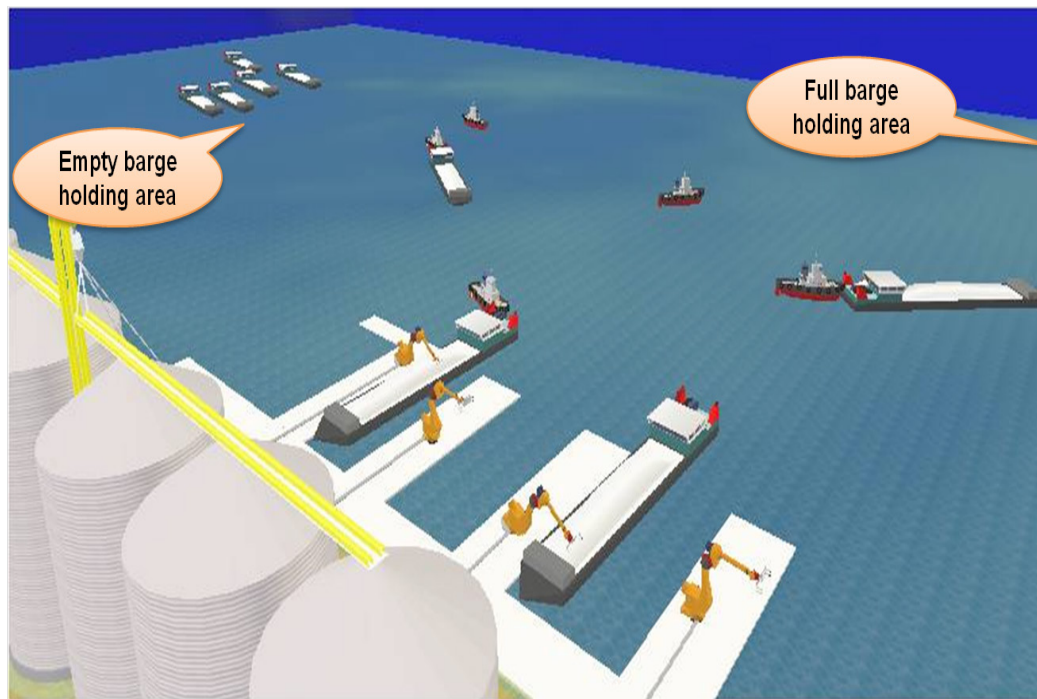


Figure 14.9 Barge operations

At the dock, time is required to connect the barge to the chute and then to disconnect it before it is free to be taken out. The times (in minutes) is

- Connection—Triangularly distributed—min: 6; max: 15; most likely: 9
- Disconnection—Triangularly distributed—min: 4; max: 8; most likely: 6

The dock operates 24 hours a day. The average number of barges arriving at the empty barge holding area are as follows. Assume Poisson-distributed arrival rates.

Time period	Avg. no. of barges per hour
00:00 – 07:00	1.0
07:00 – 15:00	3.5
15:00 – 21:00	5.0
21:00 – 24:00	1.5

The barges have the following capacities:

Capacity (tons)	Percentage
400	15
600	20
800	15
1000	40
1200	10

The distances in the operations area are as follows:

From/To	Distance (miles)
Empty barge area to any dock	1.0
Any dock to Full barge area	1.3
Between docks	0.2
Full barge area to Empty barge area	1.5

Assume the tug boats operate at an average speed of 5 miles/hr.

Expected Results

Develop a simulation model and analyze loading operations in terms of the number of barges awaiting loading, their wait, the number of tug boats used, and the tons of wheat shipped per day. Run the simulation for a 10-day period. Draw an OFD of the system first.

Modeling and Analysis Issues:

- What time unit should be the basis of the simulation?
- What level of detail will be used? What assumptions are made?
- What variables should be used and where should data be kept?
- Should fluid be used to represent the grain?
- What can represent a barge? Does it actually have to be “loaded”?
- How will the barge arrival schedule be simulated?
- How will the barges get to and from the docks?
- How will the travel time be simulated?
- What are the performance metrics?

Chapter 14 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. What is the basic unit of measure that is commonly used with materials that are considered “fluids”? Provide two examples of specific measures.
3. Name two materials that are not liquid that can be considered fluid for a simulation.
4. Describe three operating systems that might use fluid objects; describe why they would be used.

5. Describe the difference in how states change between a continuous (or fluid) flow simulation and a discrete-event simulation.
6. Describe how you might simulate a liquid flow using normal discrete simulation objects rather than fluid objects.
7. Define a “tick” time unit.
8. Describe the difference between a fluid blender object and a fluid mixer object.
9. Describe the difference in the operation of ports on a fluid object (e.g., a pipe) and a discrete object (e.g., a processor).
10. Describe operating systems where you might use a fluid-to-item object and an item-to-fluid object.
11. Select three fluid objects and identify their analogous discrete objects.

Chapter

15

Simulating Production Schedules

In today's economy, few manufacturing operations constantly produce the same product. The drive toward flexible systems means that a single production line has the capability to produce a number of products. The order in which products are produced can greatly affect the operating performance of the line. This is especially true if variables such as the setup time between products, production rate, and machine reliability vary by product.

The production schedule is often dictated by a marketing or business unit based on actual or anticipated demands. Even if the production schedule is generated by an enterprise-level optimization application, the schedule may not necessarily be optimal for the local production line. Simulation is used to validate the impact of a production schedule on specific manufacturing operations. *FlexSim* has two objects that were specifically developed to simplify the simulation of production schedule: the system controller and the line controller. The objects are contained in the `libs_Items` folder of the `BookDownloadFiles` available from the Flexsim website in the Education area. Once the files are downloaded from the website, select Open Library in the Flexsim application on the library panel, navigate to the `libs_Items` folder and select the `ProductionControLib` icon.

Production schedules can significantly impact an operation's performance

Section 15-1 Controlling production lines

Often the scope of a manufacturing simulation will include one or more production lines. The production lines are controlled by a schedule which indicates what products are to be produced, what equipment will be used, and in what order the production should take place. Simulating such an environment would normally require

a large amount of custom coding to take into account all the operating variables. In *FlexSim*, the production scheduling objects (the line controller and system controller) eliminate the need for that custom code, simplifying the process dramatically.

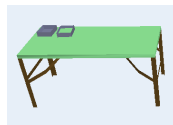
For simulation purposes, the production environment is defined as follows:

Line: A group of equipment that produces a single output (product) at a time

System: One or more lines that are scheduled as a group

FlexSim utilizes two objects to manage operations:

- System Controller



- Contains the schedule of product to be produced
- Controls operations for one or more lines
- Makes assignments to line controllers attached to it

- Line Controller



- Accepts assignments from a system controller
- Manages the activities of a single group of objects to produce a given output

Consider the example shown in Figure 15.1 of a simple production line. In this example, material comes to two cutting machines that stamp out parts. The parts continue on a conveyor to a queue before going into the painting machine. The parts are then placed in a finished-parts bin. A system controller provides the schedule and the line controller manages the equipment.

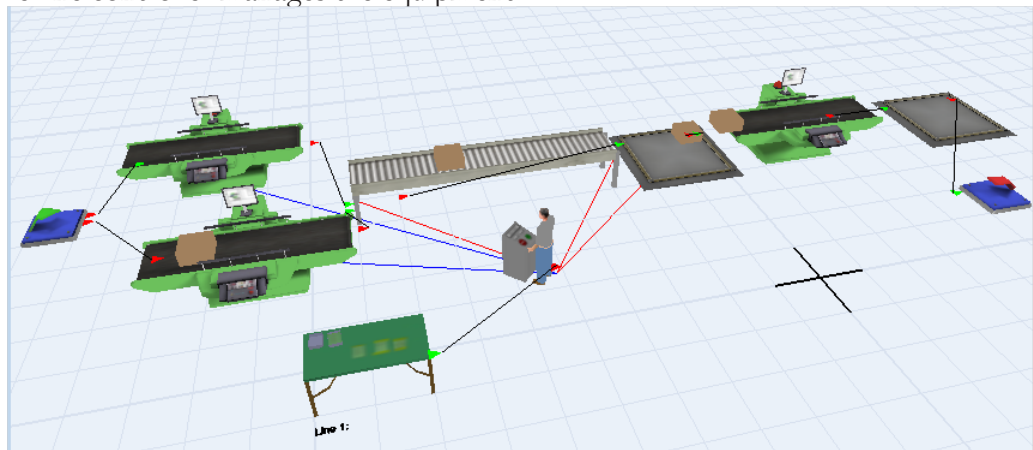


Figure 15.1 Simple Production Line Example

Section 15-2 system controller

As the repository of the production schedules for a number of lines, the system controller assigns production runs to a line and communicates information about the production to individual line controllers.

Production
is controlled
by a system
controller
managing
individual line
controllers

As shown in Figure 15.2, the main interface for the system controller holds the master schedule for all the lines. Note that the first four columns in the table

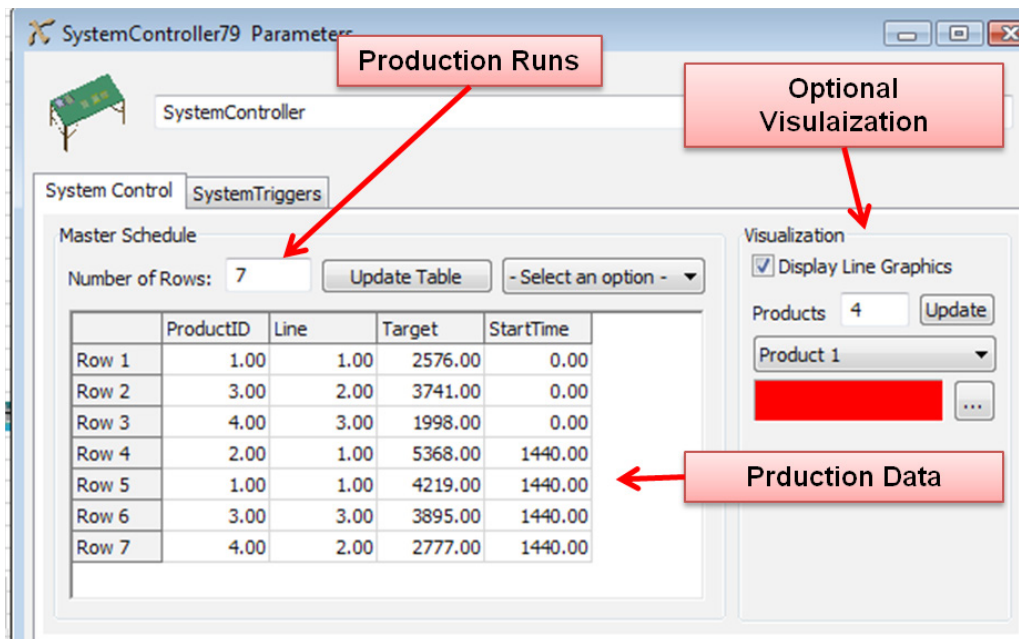


Figure 15.2 System controller main interface

(ProductID, Line, Target and StartTime) are required. The schedule items include the following:

- Number of Rows: Each row is a separate production run. The number of production runs for the entire system is entered and then updated using the Update Table button.
- Product ID: A numerical value that represents the product.
- Line: Denotes the line where the product will be produced
 - The line number corresponds to the input port number on the system controller to which the line controller is attached.
 - The line controller is connected by an A-connection, a regular object connection, from the system controller.
- Target: The number of items to be produced by the line for the run
- StartTime: The earliest time the run can start
 - Based on units of time chosen for the simulation
 - All runs with the same StartTime will run on their assigned line in the order denoted in the table.

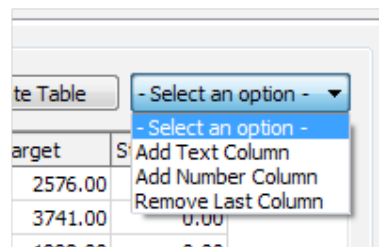


Figure 15.3 Option drop-down

The production schedule data is stored on the system controller

- Select an option menu: As shown in Figure 15.3, allows more columns to be added to the table, thus providing more information about each run
- Visualization: An optional choice; a different color can be assigned to each product.

The entries in the table do not have to be in any particular order—the system controller sorts the list each time a line becomes available to find the next run.

The SystemTriggers tab contains the typical triggers that provide a place to add custom logic with two additions :

- *OnLineAvailable* executes when a line controller reports that it is available for another run assignment. It can be used to dynamically modify line assignments.
- *OnStartTime* executes when an assignment is made to a line.

Section 15-3 Line controller

The line controller takes an assignment from the system controller and manages the equipment that makes up the production line. It is the repository for information about the equipment and the products. The information is contained on two of the objects tabs (LineParameters and Product Data).

The line controller manages all the equipment in a single production line

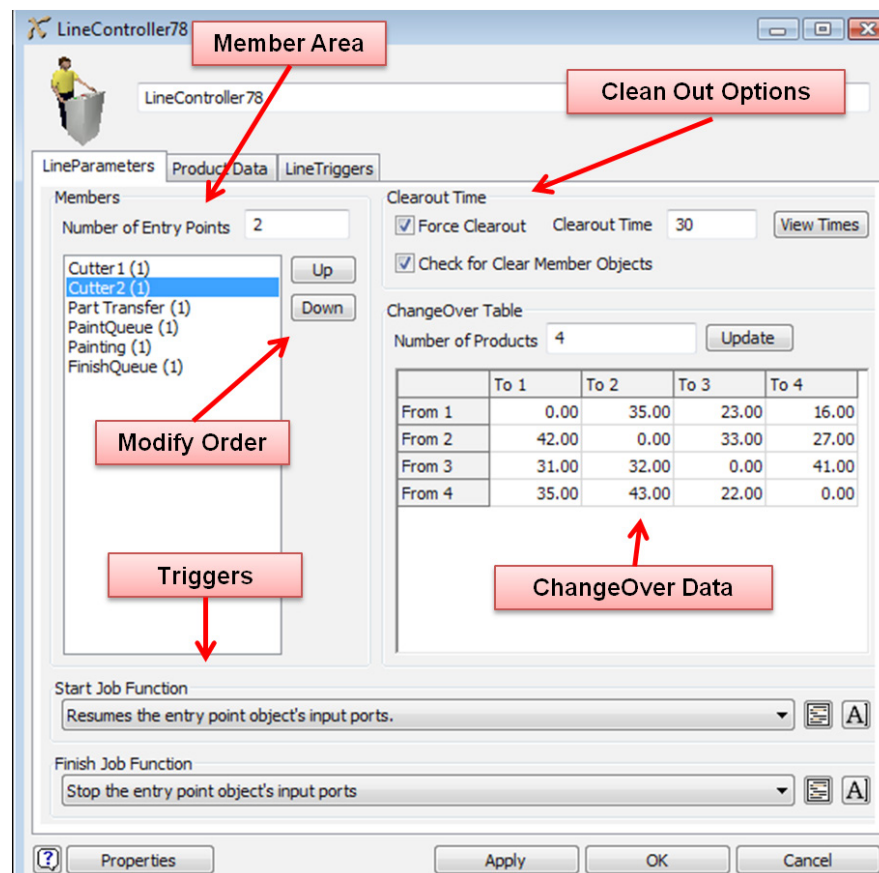


Figure 15.4 Line parameters tab

As shown in the Figure 15.4, the LineParameters tab contains basic information about the operation of the line.

- *Members*: Objects in the production line; connected by an A-connection from the line controller to the object
- *Entry point*: Identifies the object at the beginning of the line so that the input ports can be opened or closed to start or stop the line; the number of entry points is the number of objects controlled as entry points; only the entry points have their input points controlled.
- *Number of Entry Points*: Controls the number of entry points; connections from the controller for entry points are colored blue and other connections are colored red.
- *Up and Down buttons*: Changes the order of the entries in the Members table
- *ChangeOver Table*: Values represent a delay time between the production of two products; time is in the chosen units for the simulation.
- *Cleanout Time*: Delay time during which the controller waits for a product to clear out of the line before reporting that the line is available. If selected, any items remaining in the line can be destroyed, and the line controller will check for an empty line before starting a new run.
- Triggers are provided at the bottom of the tab for use at the start or end of a run.
 - *Start Job* Function by default opens the entry points.
 - *Finish Job* Function by default closes the entry points.

The Product Data tab, as shown in Figure 15.5, contains information for dynamically changing member variables depending on the product being run.

LineController13 Parameters

LineController13

LineParameters | **Product Data** | LineTriggers

Number of Variables: 3 Number of Products: 4

	Variable1	Variable2	Variable3
ObjectType	Processor	Processor	Processor
ObjectName	Cutter1	Cutter2	Painting
NodePath	>labels/CycleTime1	>labels/CycleTime2	>variables/maxcontent
Product1	3.20	3.20	2.00
Product2	4.00	4.00	1.00
Product3	1.50	1.50	3.00
Product4	2.20	2.20	3.00

Figure 15.5 Product data tab

The line controller also maintains data about the products being produced on its line

The table is sized by entering the number of variables to be used and the number of products; clicking the Update Table button applies these values.

When a product is run on the line, the variables are updated on the objects. In the case shown in Figure 15.5, the *s* label on the two cutters and the number of parts in the painting machine are updated.

To define the location of the variable to be changed, click any of the top three rows under the variable. When a row is clicked, a browse button will appear; when clicked, the browse button opens a tree view. Select the node to be changed for that variable. The information is automatically transferred to the three object row cells under the variable.

Counting Production

While a run is underway and products are being produced, the line controller needs to keep count of the production totals in order to stop the line at the correct time. The reporting is achieved using the command

```
notifylinecontroller(object, controllernumber, output, 1)
```

where the object is the member sending the information (usually current); the controllernumber is the line controller it wants to report to (an object may be a member of more than one line controller); and the output is the current output from the member which is usually obtained with the `getoutput(current)` command.

The placement of the command requires some thinking about how production is measured. Such thinking is common for actual production environments. In the above example, the command could be placed in the OnExit trigger of the source; there, it would measure the number of parts going to the line. This would work if the target contains a safety margin for losses in the line or parts left when the run stops. The command could also be placed in the OnEntry trigger of the finished product queue; in this case the target would always be reached although some additional parts would also be made as the line clears out. Remember that the items counted have to be in the same units as the target.

The LineTriggers tab contains triggers for OnReset and OnMessage as well as four triggers unique to the Line Controller:

- *OnJobStart*: Executes logic when the production run actually starts; often used to notify other lines or update statistics
- *OnChangeOverComplete*: Executes after the job starts but before the entry points are opened; may be used to execute some additional logic or checks before the line starts
- *OnTargetReached*: Executes when the production target is reached; may be used to update statistics and for other logic

It's important to determine the best location to count produced product on a line

- OnClear: Executes when the clean out time is finished and the line is clear of material

There are two additional functions, located at the bottom of the main tab of the object:

- *Start Job Function*: Opens input for the first member object; additional logic can also be applied at this time.
- *Finish Job Function*: Closes input for the first member object; additional logic can also be applied at this time.

The sequence of execution for the triggers and functions is as follows:

- | | | |
|--------------------------------|---|------------------------|
| • OnJobStart trigger | } | Same calculation cycle |
| • OnChangeOverComplete trigger | | |
| • Start Job function | | |
| • Finish job function | } | Same calculation cycle |
| • OnTargetReached trigger | | |
| • OnClear trigger | | |

Triggers on the line controller are used to specify detailed logic about the line's operation

Exercise 15-1 Custom Shapes, Inc

Background

Business customers have long gone to Custom Shapes when they need custom plastic novelty pieces with their own logo or other art work. The Custom Shapes sales team stresses how fast they can produce orders and guarantees that an order will be produced the next business day in the exact order that they are received.

To meet market demands, the plant has worked to create a true pull production system for the custom orders. Orders received during one day are organized by time received and a schedule is setup for the next day's production. The plant currently has three stamping lines.

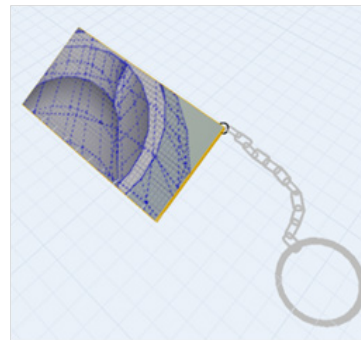


Figure 15.6 Custom shapes

The marketing department wants to expand from the smaller custom orders to larger, longer run orders for the military. Corporate management is thinking of building a new line at another facility. To try to keep the production at the plant, the engineering department has come up with a plan to increase their custom production capacity and take on the new orders by modifying their existing three lines.

Problem statement

Will the engineering department's plan to increase capacity work?

Operating data

As shown in Figure 15.7, the plant currently runs three lines that can produce five types of items. The operation of each line is the same. The stamping machines each run at 30 pieces/min for a combined rate of 90 pieces/minute.

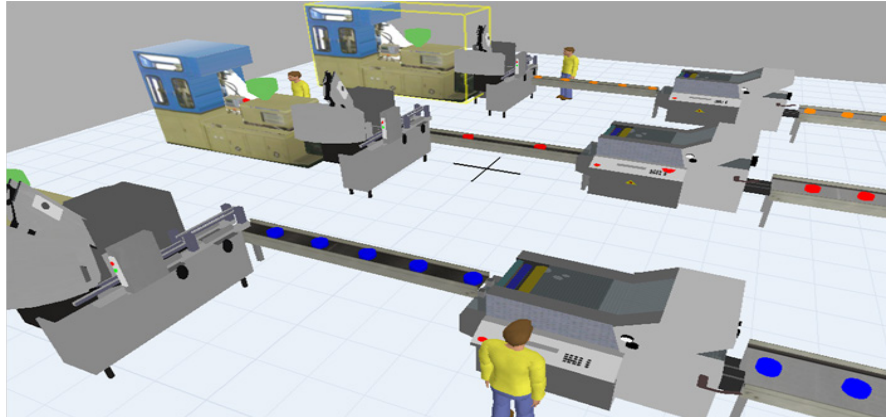


Figure 15.7 Custom Shapes Operations

- On each line plastic blanks are fed into a forming machine at a rate of 60/minute. Once formed, the shapes go into a surge that separates them and sends them to be engraved.
- From the surge they are picked up by an in-feed conveyor that sends them to the engraver, which uses a laser to print the custom design on the blank.
- The engraver is designed to run at 51 pieces/minute but can have printer errors. The loss percent at the engraver varies by the type of product.

The in-feed conveyor from the surge to the engraver is 5 feet long and has a capacity of 10 units. The surge itself has a capacity of 75 units.

Currently, all three lines are set up the same. The following table shows the five base products and the two new ones along with their loss factors.

Product	Average loss percent
Small key chain	2
Large key chain	3
Golf ball marker	4
Letter opener	1
Luggage tag	2
ID Badge (new)	3
Equipment tag (new)	2

The plant's plan is to modify two of their three lines by using two stamping machines on each of those lines which run at a rate of 24 items/min each.

The third line would remain at 60 parts/minute. In this way the two modified lines would produce a total of 96 units/minute (4x24), which would exceed the total for the original three lines. This would allow the third line to be dedicated for the new products.

Each line requires refitting for each product size. If the same size is run only the engraver has to be changed, a process that takes 2 minutes. All other product changes require 18 minutes to set up. The change times are shown in the next table, where the time is given in seconds.

From/to	1	2	3	4	5	6	7
1	120	1080	1080	1080	1080	1080	1080
2	1080	120	1080	1080	1080	1080	1080
3	1080	1080	120	1080	1080	1080	1080
4	1080	1080	1080	120	1080	1080	1080
5	1080	1080	1080	1080	120	1080	1080
6	1080	1080	1080	1080	1080	120	1080
7	1080	1080	1080	1080	1080	1080	120

The reliability of all the forming machines and engravers can be expressed as follows (in seconds):

Equipment	Time Between Failures	Time To Repair
Stamping	exponential(0,1000)	uniform(160,250)
Engravers	exponential(0,780)	uniform(120,240)

The plant scheduling department follows the sales directive and sets the line assignments based on the previous day's orders. They see no reason not to follow the same logic for the new configuration. As a test case for the simulation, the scheduling department came up with the following schedule based on the current configuration.

Daily Schedule—current configuration			
Run	Product	Line	Run Target
1	1	1	8850
2	5	2	9500
3	3	3	8750
4	4	3	8540
5	4	1	8350
6	1	2	8550
7	2	3	7800
8	2	1	7200
9	4	2	7800
10	3	3	8600
11	3	1	8800
12	4	2	8000

The schedule with the new configuration (new products on line 3 and dual stamping machines on lines 1 and 2):

Daily Schedule—new configuration			
Run	Product	Line	Run Target
1	1	1	8850
2	5	2	9500
3	3	1	8750
4	4	2	8540
5	4	1	8350
6	1	2	8550
7	2	1	7200
8	2	2	7800
9	4	1	7800
10	3	2	8800
11	3	1	8600
12	4	2	8000
13	6	3	15525
14	7	3	16480

Expected results

- Develop an OFD for the system.
- Simulate the proposed schedule on the current line configuration.
- Simulate the full schedule with the new line configuration.
- Compare the two runs at the end of a 24-hour period.
- Discuss the efficiency of the proposed operation, indicating how the changes could be beneficial.

Modeling and analysis issues

- What time unit is best for this simulation?
- How can you check that each line is running correctly?
- Is there a way to have only one simulation but be able to change the configuration for each run?
- How can you check that the entire simulation was completed?

Chapter 15 - Review questions

1. Identify three “nuggets”—the things you found to be the most interesting or most important—in the chapter.
2. Indicate two reasons why it is important to simulate production schedules.
3. Define and give examples of a line and a system in the context of production scheduling.
4. Name three machine characteristics that are important to simulation and that may have to be changed each time a product is scheduled for a line.
5. Define the major functions that are carried out by the line controller.
6. Steel bolts are manufactured and sent to bins in a packaging area. Each bin contains a different size bolt. From the bin they are placed into bags and the bags are placed into cartons. Incoming orders for bolts request a particular size, number of cartons, and number of bolts in a bag. The orders are scheduled to run in the packaging area.
 - a. List the machine variables that may have to be set for each area.
 - b. Define the entry point for the line.
 - c. Indicate where you would measure the output in order to know when to stop the system when an order is complete. Discuss the operational implications for your choice.

Appendix

Appendix for Chapter 3

This Appendix describes *FlexSim*'s general toolbars and how to move around in the simulation environment using a mouse.

Section 1 General toolbars

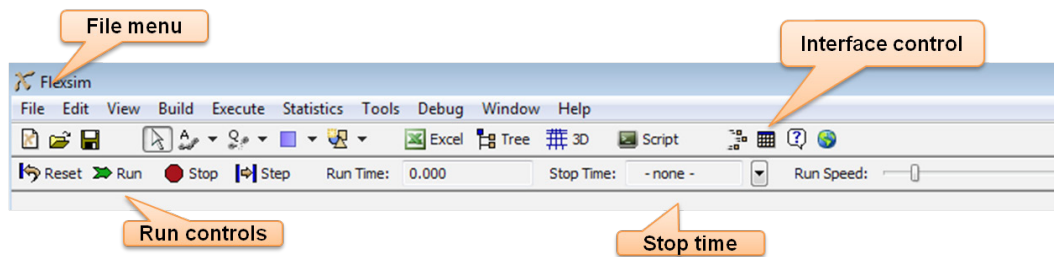


Figure A.3.1 *FlexSim* tool bars

Most of the controls used for defining and executing the simulations that were built for an Occasional User are contained in a custom-built user interface; however, some additional portions of the upper *FlexSim* tool bars, as shown in Figure A.3.1, may be of interest.

- *File menu*: The menu for file control is consistent with standard computer application practice.
- *Interface control*: Clicking the interface control icon will bring the custom interface back on the screen.
- *Run controls*: The run controls reset, run, stop, or step through the simulation. These controls are often duplicated on a custom interface.
- *Run time*: As shown in Figure A.3.2, run time displays the current simulation time. Note that the clock has no units associated with it. Actual units are defined by the user. Most often, a simulation time unit is chosen to represent one second or one minute in real time.

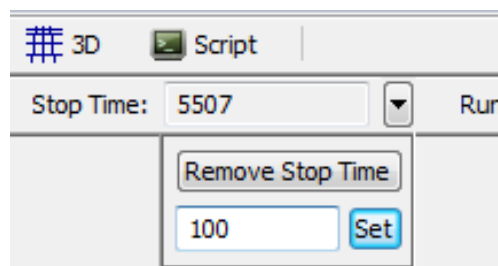


Figure A.3.2 Stop time field

Simulations are often run to a particular point in time that is used to compare various scenarios.

- *Stop Time:* The stop time field displays the time at which the model will stop running—as if the Stop button were pressed. A value of No Stop Time (the default setting) indicates that the model will run indefinitely, or until it runs out of events to process. Click the down arrow next to the stop time to set a different stop time.
- *Speed Control:* the speed control slider defines the number of simulation time units that *FlexSim* will try to calculate per second of real time. The actual result may fall short of this value if the model requires too much processing at each event. The speed value displays the value set by the slider, but it can also be edited directly by clicking the down arrow next to the speed value.

Once *FlexSim* is started, open the specific model file for the exercise by either using the File drop-down menu in the upper left corner or the Open icon just below it. Navigate to the appropriate folder to open the simulation model.

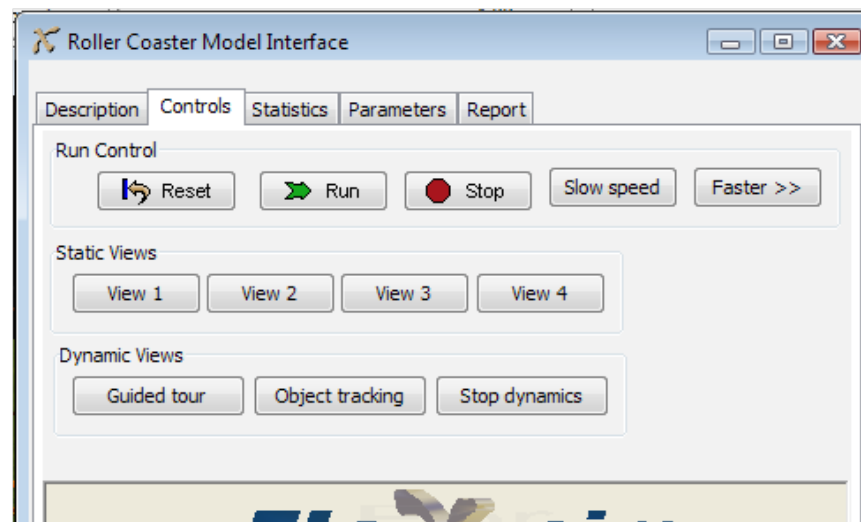


Figure A.3.3 Model controls tab

When a model opens, a model control GUI (Graphical User Interface), similar to Figure A.3.3, appears on the screen. This GUI has a number of tabs associated with it. The default tab that appears with the GUI is a very short description of the model. In some cases they duplicate controls on the simulation control bar.

The Controls tab is used to run the model as well as slow down or speed up the simulation. Rather than navigating with a mouse, different model views can be accessed by clicking on the Static View choices. Dynamic Views are provided by pre-built Fly Paths through the model. The smoothness of the fly path depends on the computer's speed, memory, and graphics card.

Buttons on the Controls tab include the following:

- *Reset:* Resets initial events and states. Before running a simulation, the initial events for the model and the initial states for the objects need to be reset;

Specialized user interfaces or GUIs are often included in simulation models for use by those not familiar with simulation or the specific software.

Operating a simulation is simplified by use of the run control buttons on the simulation Control Tab.

therefore, the reset button needs to be pressed before running a model from the beginning.

- *Run*: Begins execution of the model events. The simulation clock will advance until the model is stopped.
- *Stop*: Ceases execution of the model while it is running. It completes the current calculation cycle and updates the status of all objects in the model for that point in time. The model is not reset and it can be stopped or started again from the exact point in time that it was stopped.
- *Slow Speed*: Sets a slow speed for the model execution, while *Faster* increases the simulation in steps. The speed can also be controlled using the slider bar in the upper tool bar.

Section 2 Viewing the simulation using a mouse

Examining various parts of the simulations can be achieved by selecting the different View buttons on the Control tab; however, the view can be changed manually using a mouse.

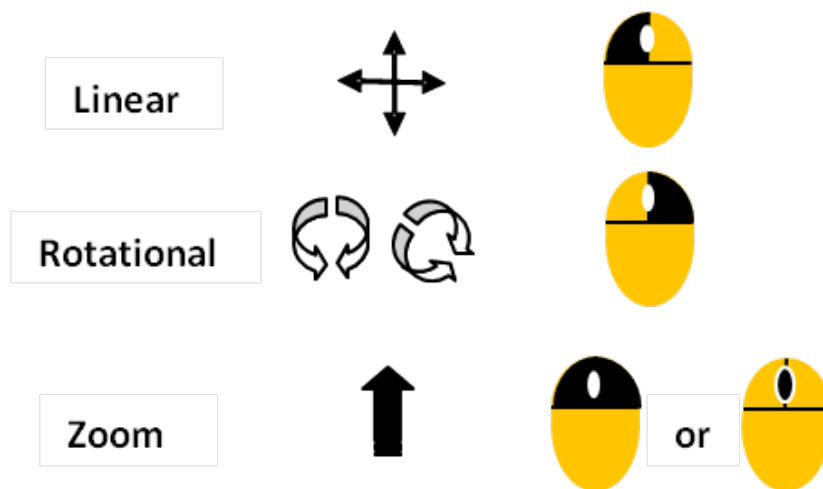


Figure A.3.4 Moving around in the 3D environment

The *FlexSim* visual environment is three-dimensional. Looking closely at a simulation while it is running is a good way to get a feel for the reality it is attempting to simulate. Detailed *FlexSim* navigation of the simulation environment is done using a mouse. It is recommended that a mouse with both left and right buttons and a scroll wheel be used for maximum efficiency.

To move around the simulation space in a linear direction, click in an empty area of the view with the left mouse button and drag the mouse around. To rotate the model view, click in a blank area with the right mouse button and drag the mouse around. To zoom out or in, use the mouse wheel or hold both left and right mouse buttons down and drag the mouse.

Moving around the simulation surface involves motion in three dimensions.

If you get lost moving around in the 3D space, right click the background (grid) and select View and then Reset View. Another option is to close the current view window using the **x** in the upper right corner of the simulation screen, then click on the 3D icon in the upper toolbar.

Should you close the model control GUI window, click on the interface control icon in the upper tool bar to open it up again.

Appendix for Chapter 5

This Appendix provides a general template for documenting a simulation modeling and analysis project. It also provides an example of Parts I and II of the general template for Exercise 5-1, Fister's Foods.

Section 1 Simulation Project Template

<<Simulation project name>>

Date modified:

Current simulation file name:

Simulation software used:

Key Words:

Part I

Problem Definition

Background (*include project/problem history, reason for simulation, justification*)

Objectives

Key Performance Measures

Key Decision Variables

Simulation Scope

Boundaries

Boundary Assumptions

Operating Assumptions

Part II

Operational Description

System Description

Supporting diagrams, photos, and historical data *(include here, reference, or attach)*

Special logic or other considerations to be included

Conceptual model / Object Flow Diagram (OFD)

REFERENCES *(for supporting documents)*

No.	Name	Location	Type	Date/ Version	Comments

Part III

Simulation Implementation

Simulation basic units of measure

1 unit of simulation time =

1 unit simulation distance =

Abbreviations and acronyms *(include color coding)*

Modeling simplifications/ assumptions

Flowitems

Flowitem name	Label name	Label type	Description / Value

Fixed resources – basic properties

Object name	Description	Capacity	Downtime

Fixed resources - labels

Object name	Label name	Label type	Description / Value

Fixed resources – Operation details

Object name	Setup time	Process time	Notes

Task executers (mobile resources) – Operation details

Object name	Capacity	Speed	Acceleration	Deceleration	Comments

Added logic on objects

Object name	Operating Parameters	Flow logic	Trigger Logic	Description

Fluid objects

Base units of flow: 1 <unit> per <time>

Tick value = 1 <time>

Object name	Description	Capacity	Port rates			
			Obj In	Obj Out	Port In	Port Out

Visual Tools

Object name	Description

Network Node Diagram (*show all objects connected to the network and network rules (e.g. one-way)*)

Global tables (*include tables here or attach*)

Random number stream assignments

Stream number	Object where used	Use

Custom objects (*list and describe those developed for this simulation or brought in from other libraries*)

User Interfaces (*describe how users interact with the simulation model*)

Simulation operation

Input variables

Output and report

Model views (*provide screen shots of simulation*)

Input data analysis

Simulation model version history

Changes from version xxx (<date>)

Part IV

Results

Model Validation

Validation plan

Validation results

Analysis questions

Definition of performance measures

Description of analysis process/methodology

Tactical Information

Initial conditions

Run length and basis

Number of replications and basis

Length of warm-up period and basis

Experiments

	Variables			Performance measures		
	V1name	V2name	V3name	PM1	PM2	PM3
Scenario	Settings			Values		

Analysis from simulation experiments

Conclusions and recommendations

Section 2 Fister's Foods exercise

Completed Parts I and II of template

Project Name: Shipping Area Capacity Analysis

Date modified: September 22, 2010

Current simulation file name: Not started **date modified:**

Simulation software used: *FlexSim* **software version:** V5.02

Key Words: shipping, capacity

Part I

Problem Definition

Background

Business is increasing and there is concern that the shipping area can't handle the higher workload without compromising the safety of the food. A number of proposals have been mentioned which are competing for funding. Simulation is being used to determine the operating capacity of the shipping area under the present and proposed conditions.

Objectives

- Determine if the current operation of the shipping area can handle the increased order rate.
- If not, recommend how the shipping area operations should be modified considering the maximum throughput and the costs involved.

Key Performance Measures

- Amount of time a frozen case remains in the shipping area
- Capacity estimate for any recommended changes

Key Decision Variables

- Staffing levels
- Degree of automation

Simulation Scope

Boundaries

- The simulation will include operations from the time a case enters the shipping area to the time the case is placed on a delivery truck.

Boundary Assumptions

- Cases continuously enter the shipping area at the specified rate.
- Trucks are always available to receive pallets

Operating Assumptions

- Operators and fork trucks are always available
- Empty pallets are always available
- Dry ice is always available

Part II

Operational Description

System Description

Food packages are placed in cases in the processing area of the plant and moved by conveyor to the shipping area. When they reach the shipping area they are transferred from the processing conveyor to the shipping area conveyor by a robot. The conveyor moves the cases to the packaging station, where an operator takes them to the packaging machine, which adds dry ice and seals the case.

Packages are coded for two shipping distances and the operator must set up the packaging machine to ensure the correct amount of dry ice is added. When packed and sealed, the operator places the case on a pallet. When the pallet is filled, an overhead crane moves the loaded pallet to a pickup area. The operator moves an empty pallet in place to continue loading.

Supporting diagrams, photos, and historical data

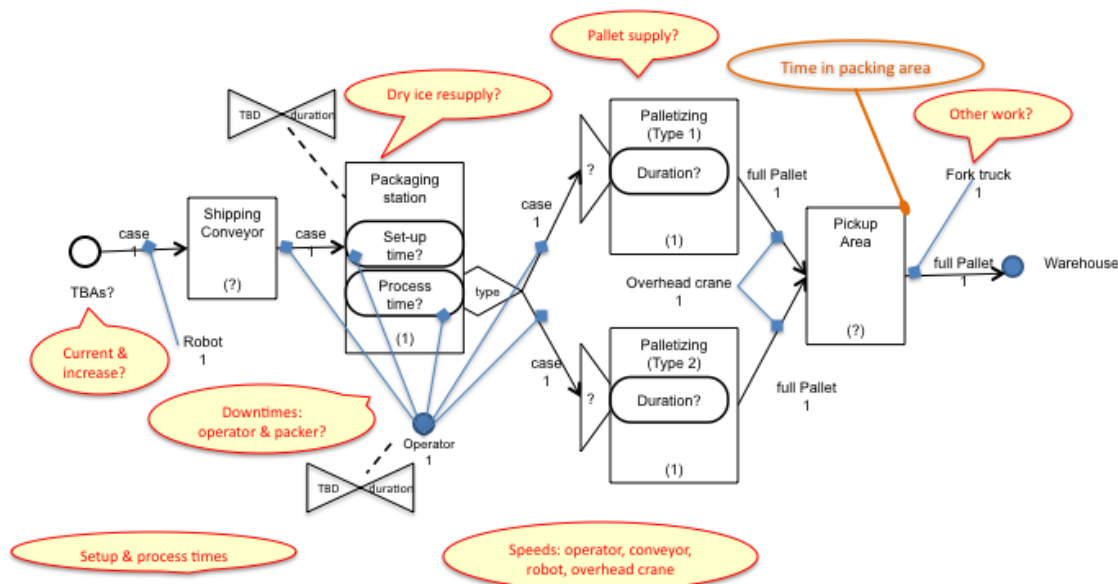
See reference list

Special logic or other considerations to be included

The packing machine must be able to change operating times based on the case code.

Conceptual model /Object Flow Diagram (OFD) REFERENCES *(for supporting documents)*

No.	Name	Location	Type	Date/Version	Comments
-----	------	----------	------	--------------	----------



1	Area operation figures	Simulation, Modeling, and Analysis	Figures	Edition 1 2011	Textbook Chapter 5
2	Machine Downtime	Plant data base	PLC output reports		Data for robots
3	Production plan	Planning department	Rate projections		5-year plant plan
4	Products	Production office	Product list		Products and shipping codes
5	Costs	Financial office	Cost records		Direct and overhead costs
6	Equipment costs	Engineering	Proposals		Bid requests

Appendix for Chapter 6

This Appendix provides instructions on how to change basic view settings, describes how to modify object visuals, discusses picklists, and describes laying out a conveyor in *FlexSim*. It also provides additional information on Exercise 6-1, Johnson Pharmaceutical, and Exercise 6-2, Lucky Air.

Section 1 Modifying the basic View settings

The way that the simulation surface and objects on it are viewed can be changed using the View Settings menu. The menu is located under the View drop-down list at the top of the window. Selecting Modeling Utilities and then View Settings brings up the View Settings drop-down list.

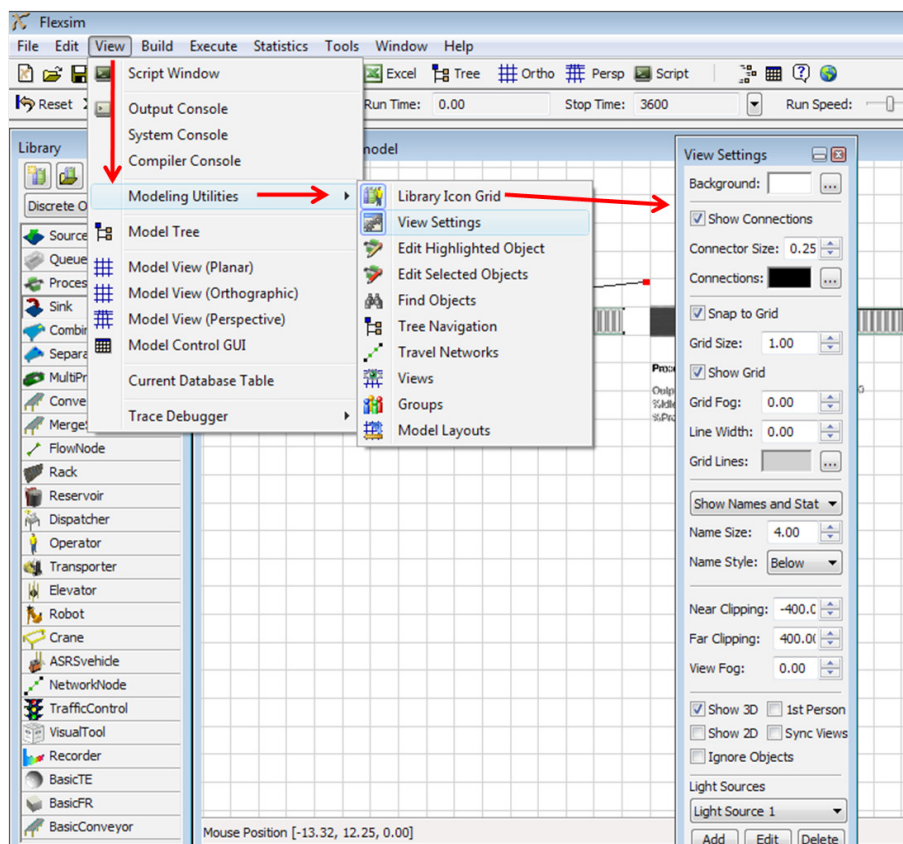


Figure A.6.1 Changing the environment with the View Settings menu

Common settings include changing the background color, showing connections, showing the grid, and using the Snap to Grid feature. The settings are established for the current simulation. New simulations will start with the default view settings.

Section 2 - Selecting and modifying object visuals

FlexSim contains numerous tools for making the simulation model building task more efficient. This section discusses the difference between *highlighting* and *selecting* objects on the simulation surface and describes how to change the appearance of an object by changing its 3D graphic.

Highlighting and Selecting an Object

Highlighting

- When an object is highlighted, a yellow box appears around the object.
- An object is highlighted by default by clicking on it with the cursor.
- Only one object may be highlighted at a time.

Selecting

- When an object is selected, a red box appears around the object.
- An object is selected by clicking on it while holding down the shift key.
- Multiple objects may be selected by holding down the shift key and dragging the cursor to cover the objects.
- A selected object may also be highlighted.
- Selected objects will move as a group.
- Objects are de-selected by holding the shift key and clicking on a blank space on the simulation surface.

Once selected, an object (or objects) can be modified by using the Edit Selected Objects menu. The menu is found by right clicking any object and selecting Modeling Utilities/Edit Selected Objects (Figure A.6.2). Alternatively, the menu can be found using the View drop-down menu at the top of the screen and selecting Modeling Utilities/Edit Selected Objects.

The menu screen changes depending on the function that is chosen. The Duplicate and Delete choices are useful on the Edit menu. The Copy From Highlighted function also is useful. Individual attributes (variable, triggers, size, labels, etc.) can be copied from the highlighted object to the selected objects. This function makes populating a number of objects with similar logic very simple.

Changing the 3D visual on an object

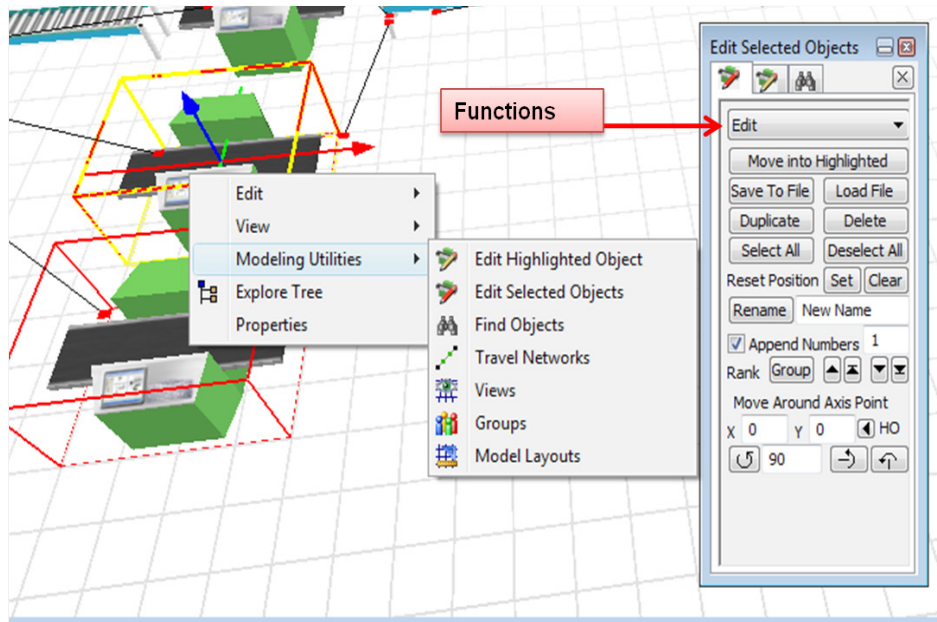


Figure A.6.2 Editing a selected object

All objects have 3D images associated with them. The visual properties of the object are changed via the General tab. As discussed previously, properties at the bottom of the tab change the size, location, and rotation of the object's 3D image.

In order to change the image itself from the General tab, click the Browse icon (...) for the 3D Shape, as shown in Figure A.6.3. and then navigate to find a new image. The fs3d folder in the *FlexSim* program folder contains a number of 3D images.

Note that a number of file formats are supported for import into *FlexSim*. The Google 3D Warehouse (sketchup.google.com/3Dwarehouse) has a large number of 3D objects that are available for use, and the .skp file type can be used in *FlexSim*. Downloaded images can be modified using the free Google *Sketchup* application. Stand-alone 3D objects can be imported by using the visual tool object. Do this by choosing Imported Shape from the Visual Display picklist on the object's Display tab.

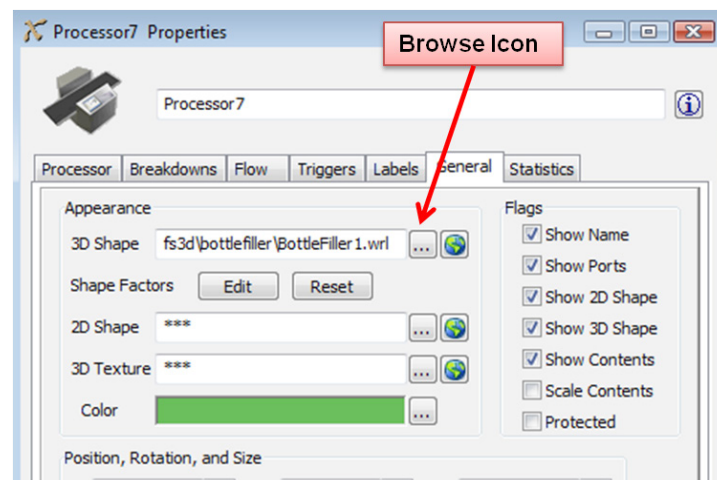


Figure A.6.3 Changing a 3D image

If a new image is being brought into a standard object (such as a processor), it is a good idea to reset the Shape Factors to 1 before importing the new shape, as shown in Figure A.6.4.

If the resulting image does not fit the yellow box around it, modify the image using by selecting the Edit button and adjusting the shape factors until the image is centered in the box. Move the edit window so that the object is visible while making the changes. Make final adjustments using the controls on the object's General tab.

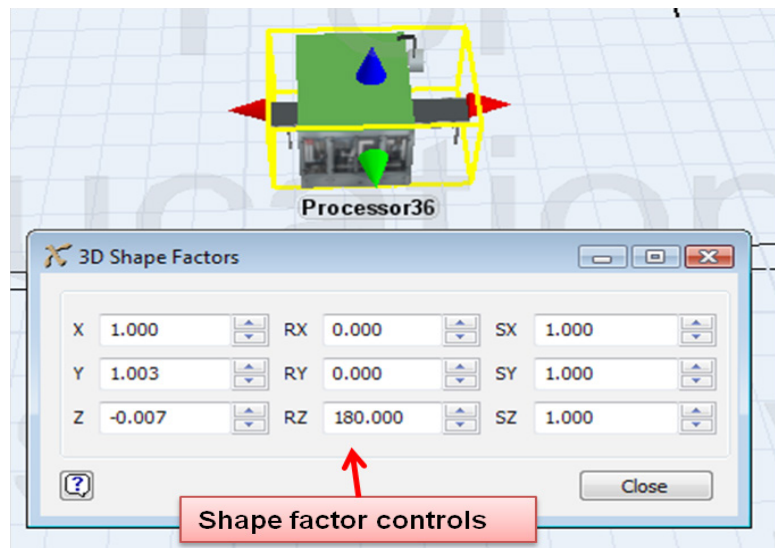


Figure A.6.4 Adjusting the image with the shape factor controls

It is good modeling practice to store all of the imported images in one folder or in the same file folder as the model.

Section 3 Picklists

Figure A.6.5 illustrates the picklist used for choosing the process time. A numerical value or other expression can be typed directly into the box. In the Figure, the drop down choice of a statistical distribution has been selected. The drop down list titled distribution provides a list of available distributions. Parameters associated with the choice may be entered directly in the boxes..

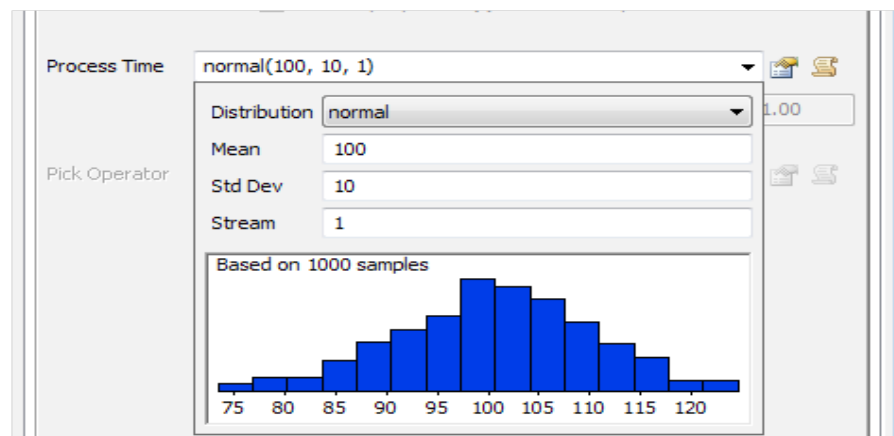


Figure A.6.5 Picklist options for processor process time

To the right of the drop-down box there are two buttons; each provides a different way to edit the selected picklist option parameters.

The first button brings up the parameter choices associated with the pick list choice. The second way to edit the picklist option is via the second button to the right of the list. It is the Code Edit button. This is a more advanced option that allows the user to directly manipulate the modeling language associated with that attribute which is discussed in Chapter 12. The Intermediate User should be able to construct models through the picklist options alone and does not have to deal with advanced techniques or scripting language.

Section 4 - Conveyor layout

Conveyors can be configured to follow complex paths in a simulation by using the Layout tab, as shown in Figure A.6.6. The complex paths can involve multiple changes in direction in all dimensions (x, y, and z). This is accomplished through the creation of multiple sections. Regardless of the number of sections used in a conveyor, it is still considered a single object.

Curved or straight sections can be added by clicking on the appropriate button. Other layout attributes change the length, rise, angle, and radius. After typing a new value, click outside the cell for it to take effect, or click on the Apply button at the

Conveyors can be structured to follow straight or curved paths.

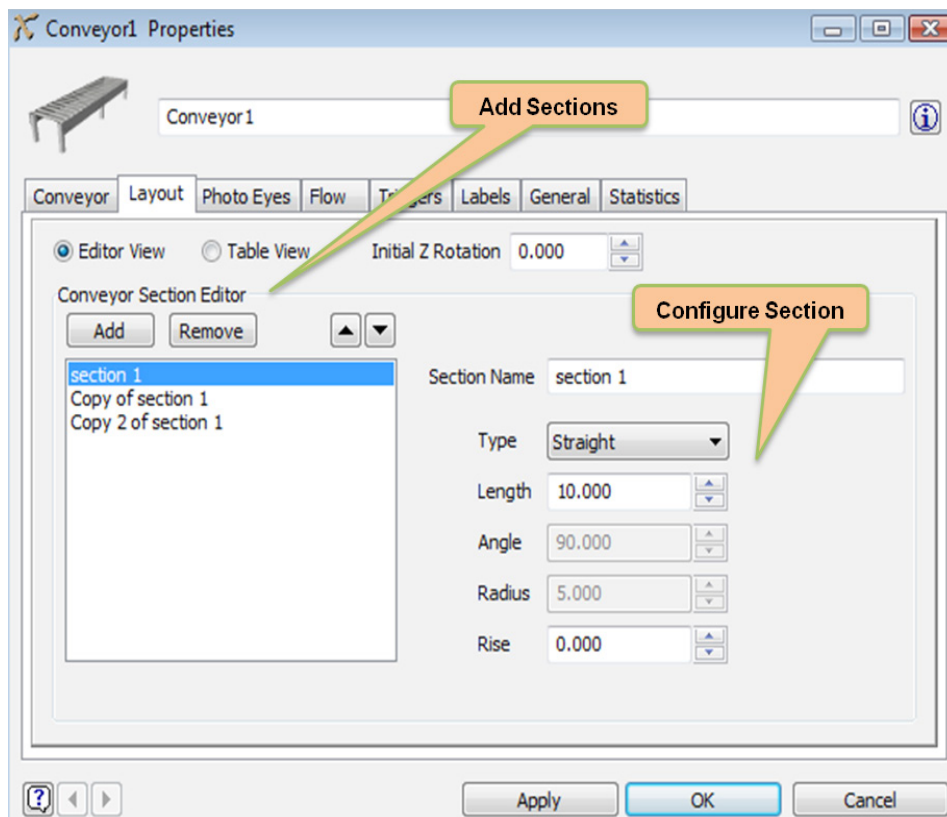


Figure A.6.6 Conveyor Layout Editor View

bottom of the window. Experiment with a single conveyor on the screen and try the various attributes. Selecting the “Table View” button provides a tabular view of the same configuration.

It is best to move the Layout window to the side while experimenting with the various attributes; this way, both the Layout window and the conveyor in the simulation window can be seen at the same time. The Initial Z rotation box indicates the visual direction of the conveyor. Move the arrows up or down to change the visual direction of the conveyor. A specific angle can also be typed in.

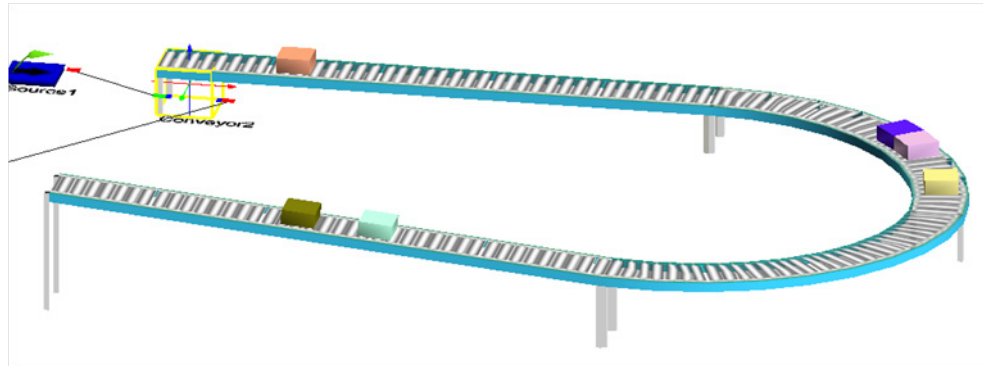


Figure A.6.7 Conveyor configured by using the Layout tab

The height of a conveyor can be changed. Usually the legs will follow the conveyor but there are times when the conveyor should be shown as operating on the ground, as shown in Figure A.6.8.

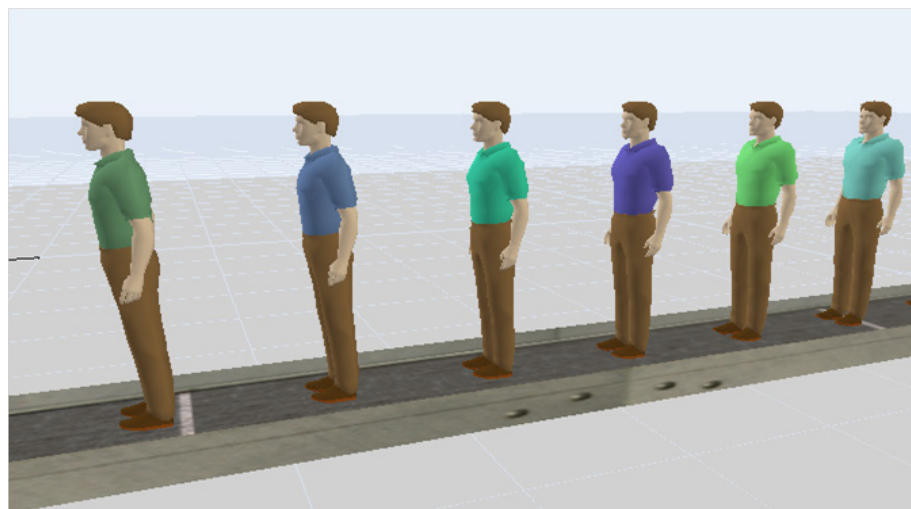


Figure A.6.8 Moving sidewalk

The basic conveyor image can be changed to resemble a moving walkway as follows:

1. In order to remove the legs from the conveyor graphic, check the “leg base relative to the conveyor” box found on the Layout tab.

2. On the General tab for the conveyor object, set SZ to a negative number to bring the conveyor down to the model surface.

There are other examples of conveyor configurations in the *FlexSim* Help.

Section 5 Exercise 6-1 Johnson Pharmaceutical

Notes:

- Build the conveyors in sequence starting with the one that goes from the source to the first Test/Wrap machine.
- Add a second conveyor to go from the end of the first conveyor to the next machine, etc.
- Be sure to connect the conveyors correctly.
 - The first output connection must go to the machine.
 - The second output connection goes to the next conveyor in sequence.

Notice that about 5% of the boxes are lost; that is, not routed down one of the three Test/Wrap machines.

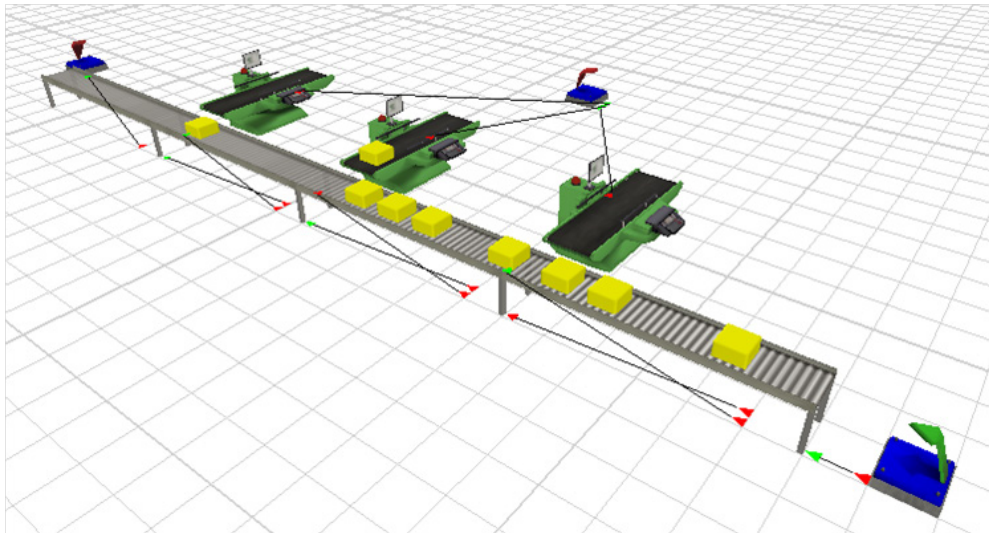


Figure A.6.9 Pharmaceutical packing line model

Section 6 Exercise 6-2 Lucky Air

Notes:

- Select the Person flowitem class from the drop-down menu on the source.

- In order to gain experience with multiple constructs and to compare their capabilities and behavior, use a different mode or object to bring each type of passenger from the source to its agent.
 - Use a queue
 - Change its shape using the General tab to be a long and narrow waiting line
 - On the main queue window select Stack Inside Queue from the dropdown menu for Item Placement.
 - Use a conveyor
 - Use a flow node
- If you assume each grid unit is 2 feet, and time is in minutes, then set the speed on the conveyor and flow node to 125. This means the passengers will move 125 grid units per time unit or 250 feet per minute (standard walking speed of about 3 miles per hour)
- Use the Statistics tab on the various objects to obtain the simulation model results.

It looks strange to see people moving on a conveyor, although a slight change in the graphics and they will look like they are on a moving sidewalk. Similarly, it looks odd that the passengers flow down a machine for the time they spend being served by an agent. Later we'll see how these can be fixed by some tweaking of the graphics; however, for now the focus is on mastering the basic modeling concepts and understanding how the different constructs work. Notice that the objects have been named using the recommended convention; that is,—using the object type abbreviation and a descriptive name.

A completed simulation project template might look like the following:

Part I

New check-in options

Background

Simulation is being used to test concepts for setting up a new check-in operation.

Objective

The objective of this modeling and analysis project is to support the design for a better check-in area for Lucky Air passengers, including layout and agent operations.

Key performance measures

- The time passengers wait to be served by a ticket agent.
- Agent work load.

Key decision variables

- Number, type, and arrangement of waiting lines in the check-in area.
- Order in which passengers are served.

Simulation scope

- Only the check-in area is considered, in other words, from the time a passenger arrives in the area until they have completed service with an agent.

Operations Assumptions

- Passengers join the correct line upon arrival and remain there until served.
- Arrival process is stationary or constant over the day.
- Three agents are always available; that is, operators are covered during breaks, equipment never fails, etc.

Part II

Operational Description

System Description

Lucky Air passengers can be classified into three main types: those who use an e-ticket, those who use a paper ticket, and those that need to purchase a ticket. Also, about 10% of Lucky Air's passengers are frequent flyers and are considered "special." Currently, Lucky Air uses three agents to serve the passengers.

Under the current system, passengers, upon arrival to the area, join one of three lines depending on the type of ticket they will use (e-ticket, paper, purchase). There is space for each waiting line to be 30 feet in length without interfering with aisles in the terminal. The agents are specialized and handle only one type of customer.

The rationale for the system is that it is more efficient to deal with each type of customer separately and not commingle them. That is, all of the faster processing times (those with e-tickets) are grouped together, all of the slower transactions (those purchasing tickets) are handled together, etc. Of course, one downside is that agents that are idle are not available to help other types of customers.



Figure A.6.10 Passenger check-in and ticket agent

Frequent flyer customers join the appropriate line just like regular customers. Passengers are processed by the agents on a first-come, first-served basis within each category.

Supporting Diagrams and Photos

A typical check-in area is shown in Figure A.6.10. No architectural drawing, process diagrams, value stream maps, etc. are available.

Special logic or other considerations to be included

None required for this simulation

Conceptual model

The object flow diagram in Figure A.6.11 provides a conceptual representation of the system, indicating key resources, both fixed and mobile, entity flows through the system, and primary performance measures.

References

There are no additional references.

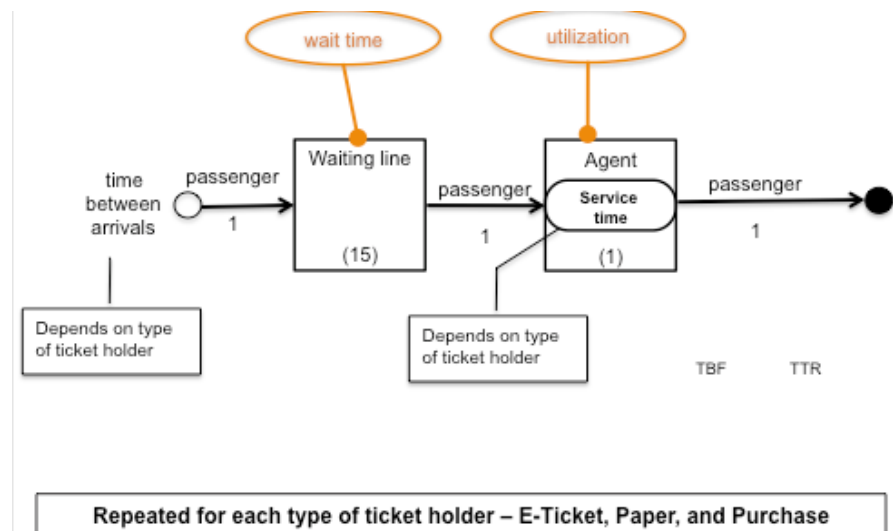


Figure A.6.11 OFD for Lucky Air

Part III

Simulation Implementation

Units of measure

1 grid unit = 2 feet

1 time unit = 1 minute

Abbreviations and acronyms

None

Modeling simplifications / assumptions:

None

Flowitems

Name	Property / LabelName	Value / LabelType	Description / Value
fi_Passenger	ItemType	numeric	1= e-ticket, 2=purchase, 3=paper
	Priority	numeric	0=Regular, 1=Special
	ServiceTime	numeric	by ItemType:
	Color	by ItemType	1=red, 2=yellow, 3=blue

Fixed Resources – basic properties

Name	Description	Capacity	TBF/TTR
qu_ETicketLine	Ticket line for e-ticket passengers	15	none
cv_PaperLine	Ticket line for passengers with paper tickets	15	none
fn_PurchaseLine	Ticket line for passengers buying tickets	15	none
pr_Agent_ETicket	Agent for e-ticket passengers	1	none
pr_Agent_Paper	Agent for paper ticket passengers	1	none
pr_Agent_Purchase	Agent for passengers purchasing tickets	1	none

Fixed resources – labels

Name	Property / LabelName	Value / LabelType	Description / Value
cv_PaperLine	speed	125	
cv_PaperLine	length	15	30 feet
fn_PurchaseLine	speed	125	
fn_PurchaseLine	length	15	30 feet

Fixed Resources – Operation details

Name	Setup Time	Process Time
pr_Agent_ETicket	none	Normal(3,1,11)
pr_Agent_Paper	none	Normal(8,3,12)
pr_Agent_Purchase	none	Normal(12,3,13)

Task executers

No task executers are used in the model.

Added logic on objects

Object name	Operating Parameters	Flow logic	Trigger Logic	Description
sc_ETicket	Interarrival time			TBA~exponential(0,5,1)
			OnExit	Set color to red
sc_Paper	Interarrival time			TBA~exponential(0,10,2)
			OnExit	Set color to yellow
sc_Purchase	Interarrival time			TBA~exponential(0,15,2)
			OnExit	Set color to blue

Fluid objects

No fluid objects are used in the model.

Visual tools

Object name	Description
vt_Time	displays current simulation time.
vt_Units	displays time and special units

Network information

No networks are used in the model.

Global tables

No global tables in the model

Time tables

No time tables are used in the model.

Random number stream assignments

Stream Number	Object where used	Use
1	sc_ETicket	IAT for E-ticket passengers
2	sc_Paper	IAT for paper ticket passengers
3	sc_Paper	IAT for purchase passengers
11	pr_Agent_ETicket	ST for E-ticket passengers
12	pr_Agent_Paper	ST for paper ticket passengers
13	pr_Agent_Purchase	ST for purchase passengers

Custom objects

No special objects are used in the model.

User interfaces

Simulation operates using standard *FlexSim* interfaces.

Model views

A model view of the base case is provided in Figure A.6.12.

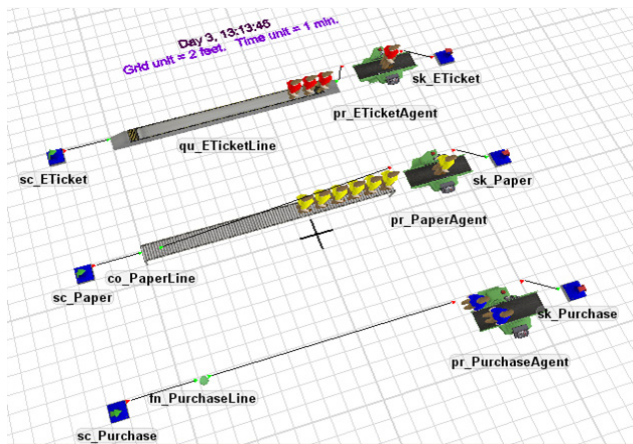


Figure A.6.12 Model view of the base case

Input data analysis

As defined in system definition.

Version History

Version 1.0

Part IV

Results

Model Validation

Validation plan

Check that expected arrivals and agents' workloads are reasonable.

Validation results

The expected number of arrivals is the average arrival rate multiplied by the simulation time. For the e-ticket passengers, the average time between arrivals is 5 minutes, which means the average arrival rate is 12 per hour; therefore, 2,016 are expected in 168 hours. Given the high variability in the exponential distribution, a result of 1,982 arrivals is reasonably close (within 2% of the expected). Overall for all passenger types, there are a total of 3,688 arrivals, compared to an expected total of 3,696 (within 0.2%).

Analysis questions

Definition of performance measures

- The average and maximum length of the lines for each agent's station
- The average wait-time for each type of customer

- The average number of customers each station services per hour (throughput)
- The average utilization for each agent

Description of analysis process / methodology

Tactical information

Initial conditions: All lines are empty and agents are idle.

Run length and basis: 168 hours

Number of replications and basis: 1

Length of warm-up period and basis: No warm-up time used.

Simulation Results

The following results were obtained for a 168-hour run of the simulation. See Figure A.6.13.

	E-Ticket (queue)	Paper ticket (conv)	Purchase ticket (flownode)	Total or average
Number of customers served	1982	1044	653	3679
Arrivals (expected (actual))	2016, 1982	1008, 1051	672, 655	3696, 3688
Throughput (customers per hour)	11.8	6.2	3.9	21.9
Average length of waiting line	0.43	1.67	1.85	1.03
Maximum length of waiting line	7	12	15	9.94
Average waiting time (minimum)	2.17 (0)	16.05 (0.12)	28.58 (0.12)	10.80
Agent utilization (%)	58.9	82.1	77.6	72.9

Figure A.6.13 Simulation results

Conclusions and recommendations

There was a definite imbalance in waiting time for the different types of passengers and in workload of the three agents. It is possible that if the agents are trained to handle any type of passenger that the times and agent workloads would improve. Further simulation analysis is recommended.

Appendix for Chapter 7

This Appendix describes how to put labels on flowitems and how to use source object triggers in *FlexSim*. It also provides additional information on Exercise 7-1, More Lucky Air, and Exercise 7-2, Even More Lucky Air.

Section 1 Putting labels on flowitems

Although you can use the default flowitems in new models, it is good practice to create flowitems that are specific to the model. This is done through the Flowitem Bin as shown in Figure A.7.1. In this case, the default Person flowitem class is selected as the base flowitem and, by selecting New Item, a copy of the base object is created so that it can be customized.

The interface to the right in the figure is used to edit the flowitem properties, such as name, size, color, etc. It is also used to add labels to the flowitem. In the case below, a numeric label named ServiceTime is attached to the flowitem.

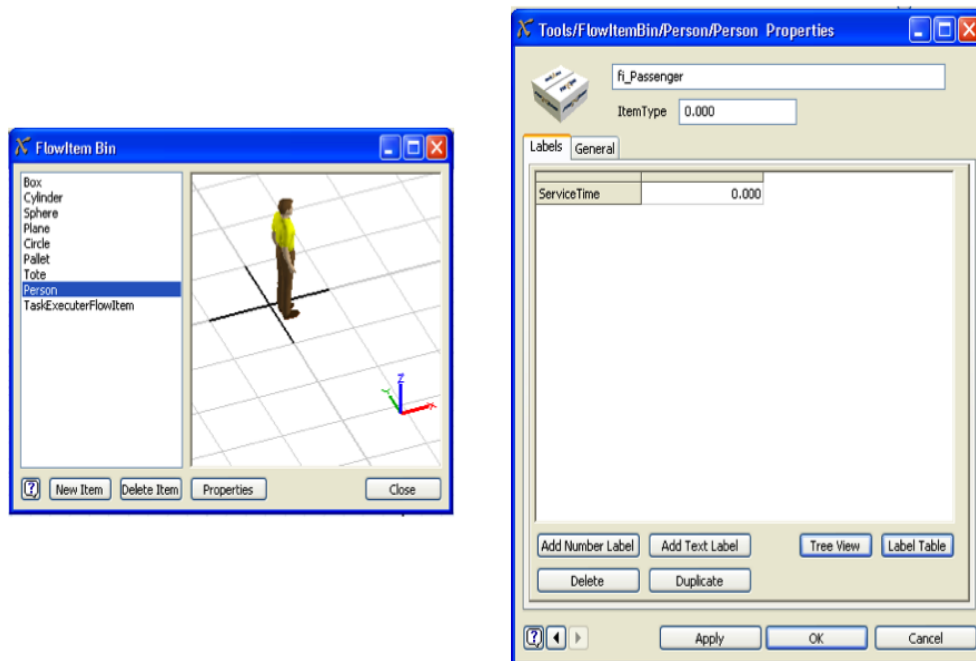


Figure A.7.1 Defining flowitems and adding labels.

Section 2 Modifying trigger logic with picklists

The desired actions that occur at a trigger event are set by the drop-down menu options for each trigger type, as shown in Figure A.7.2.

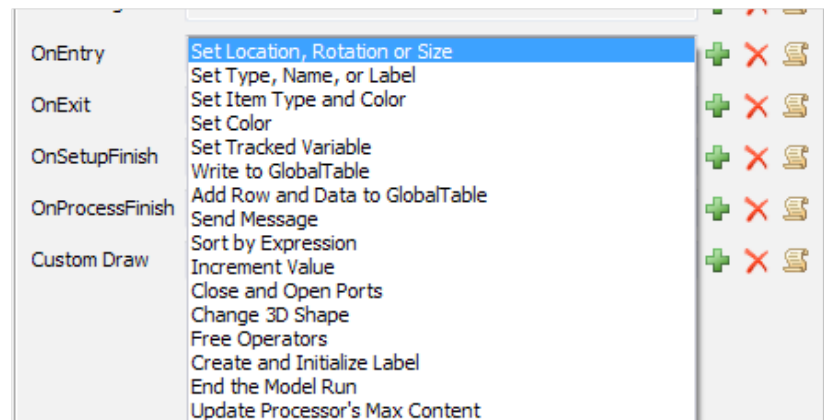


Figure A.7.2 Typical OnEntry Pick List - + icon

Trigger selections have icons associated with them. The choices will vary depending on the object.

- The + icon opens a picklist of possible logic choices. If another choice is made it will be added to the logic for the trigger. Making a selection brings up a template window to modify selection parameters as shown in Figure A.7.3.
- The X icon will remove all picklist selections.
- The Code icon opens a window of the *Flexscript* code or the Logic Builder for the logic choice. This is the place for creating custom code and is discussed in the Advanced Section of this book in the Building Custom Logic chapter.

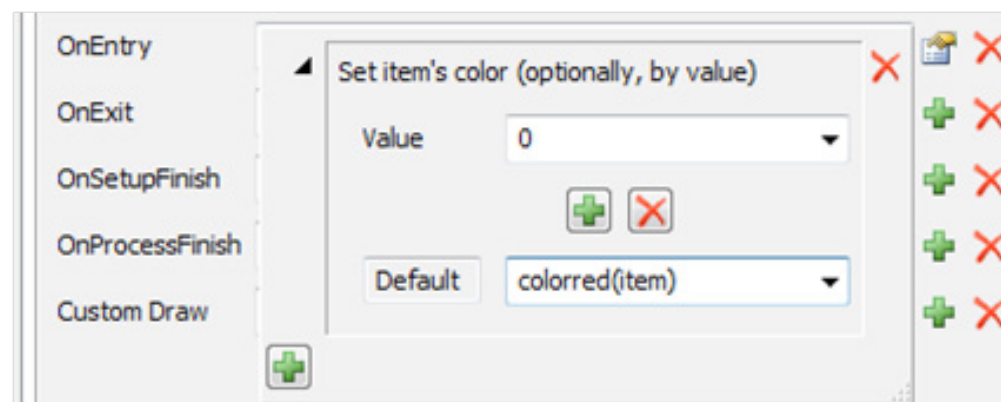


Figure A.7.3 Template view of color selection

Example using source object triggers

Note that there is a unique trigger on source objects, called OnCreation. This trigger fires when a flowitem is created by the source. Oftentimes the OnCreation and OnExit triggers fire during a single event causing the triggers to appear to fire at the same time because the flowitem exits the source when it is created; however, if the source is connected to an object with limited capacity (e.g., a queue object with maximum contents set to 10) and if the queue object is full when a flowitem is created and set to exit the source, then the source will hold the flowitem until it can be moved out. In this case, therefore, it becomes clear that the OnCreation and OnExit triggers will execute at different times.

One way to tell if flowitems are being held by a source is to look at the source's "percentage blocked" statistic. If it is greater than 0%, then at times the source has been blocked and has had to retain flowitems when they were created.

The OnCreation trigger in Figure A.7.4 does two things to the flowitem when it is created: it sets its itemtype to 1 and sets the flowitem color to red. Notice the default logic for Set Itemtype is changed from the default (randomly set the itemtype to a value between 1 and 3) to 1.

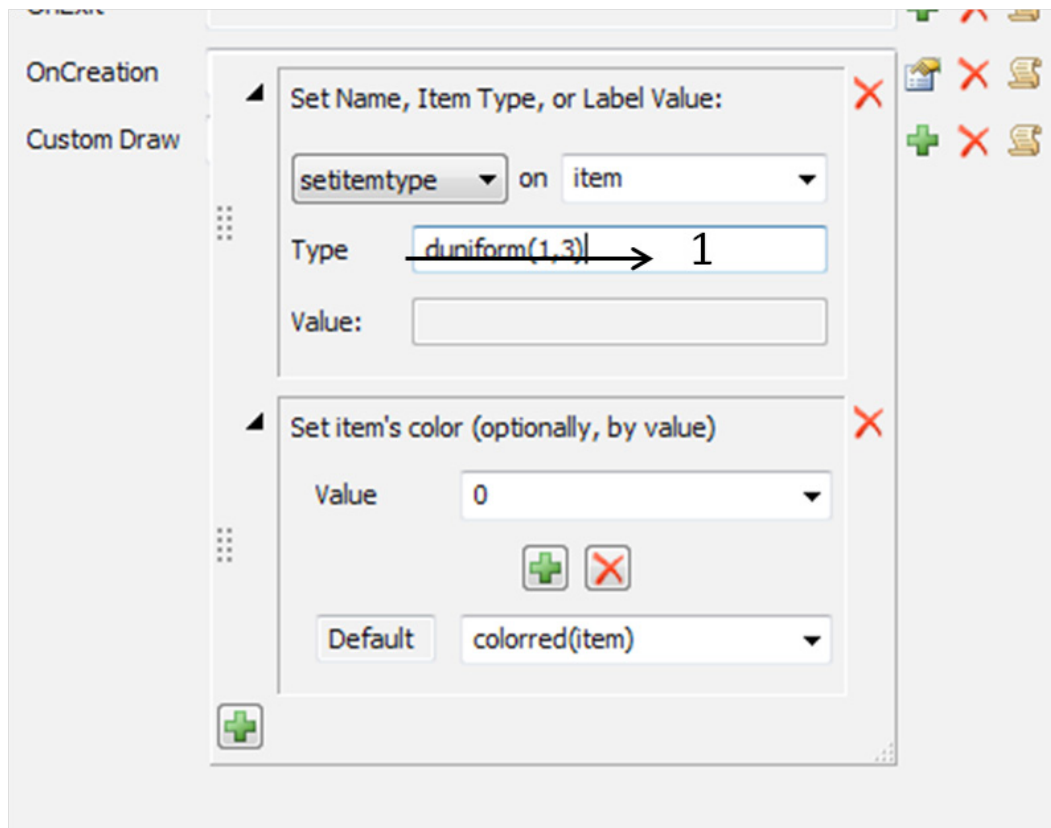


Figure A.7.4 Setting flowitem properties on the source

Section 3 - Example 7-1 More Lucky Air

In terms of objects and their relationships, the model's layout should resemble that shown in Figure A.7.5.

For this model, most of the objects are the same as in Exercise 6-2; the sources, processors representing the agents, and sinks are the same. A single line replaces the three conveyors that represented the lines for the three types of passengers.

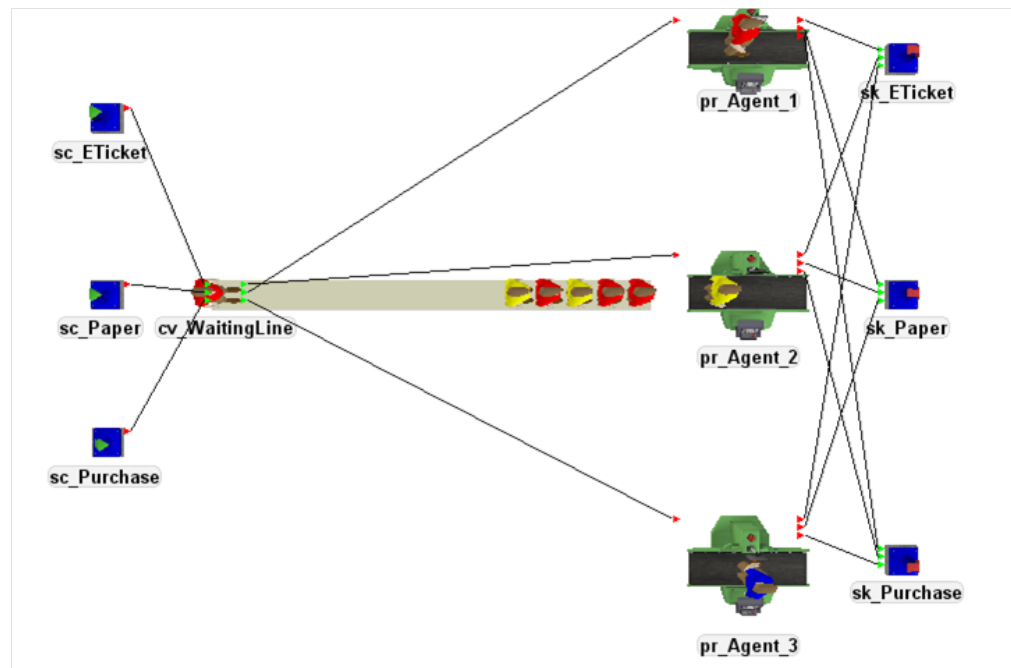


Figure A.7.5 Initial “More Lucky Air” model, Exercise 7-1

Modifying the source

Each source in the “More Lucky Air” model needs to be edited: e-ticket passengers will be itemtype 1 and colored red; passengers with paper tickets will be itemtype 2 and colored green; passengers who purchase tickets will be itemtype 3 and colored blue.

Finally, the ServiceTime label value needs to be set; that is, each flow item needs to have its service time at the agent randomly generated at the time it exits from the source. If the service time is stored as a label on the passenger, it should be stored in the appropriate label for use later in the model. The service time will be used to set the processing time at the agent.

The trigger that is used to set the label value can either be added to the OnCreation trigger actions, or the OnExit trigger can be used; it really doesn't matter. Figure A.7.6 shows the label value being set using the Create and Initialize Label picklist option in the OnExit trigger. This selection assumes that the label hasn't

been created on the flow item. When the label already exists, the Set Label selection should be used.

The template values need to be edited carefully. For example, when entering a string value for the label name, the value must be spelled correctly. Also, the string value is case sensitive; for example, `ServiceTime` is different from `Service Time` and `ServiceTime`. If the name is improperly entered, it will be ignored by FlexSim. Normally string values must be enclosed in quotation marks, “ ”, when used in writing code in Flescripts as shown in the Advanced User section of the book. In the template, as shown in Figure A.7.6, however, they should not be used.

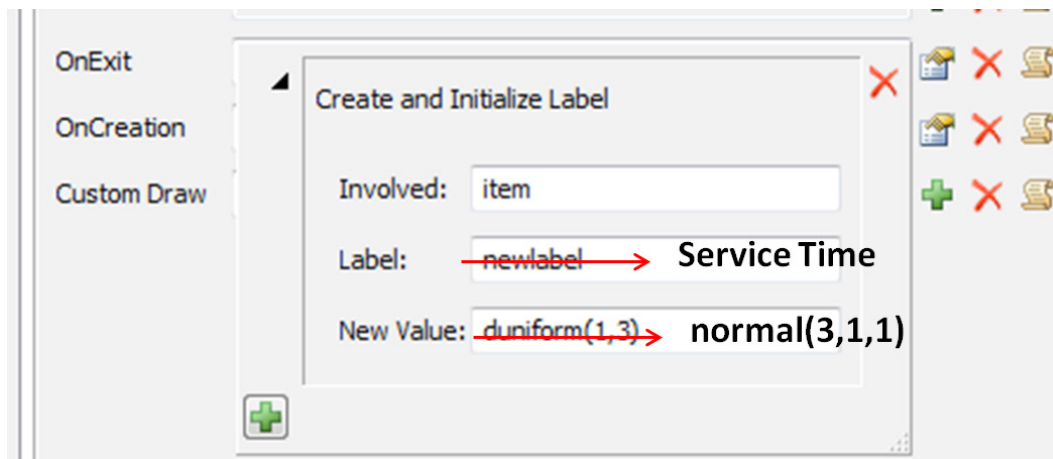


Figure A.7.6 Setting a flowitem's label at the source object

Note that the value of the label is being set by a command that obtains a random sample from the normal distribution. The three parameters in the command are the mean, standard deviation, and random number stream, respectively. The random number stream will be discussed in a later chapter.

The other two sources in the “More Lucky Air” example need to be edited to reflect the service times for the passengers that use paper tickets and purchase tickets from an agent.

Connecting passengers to the agents

When connecting the output ports of the waiting line to the agents, the connection order is important. As shown in Figure A.7.7, the model indicates that the first choice

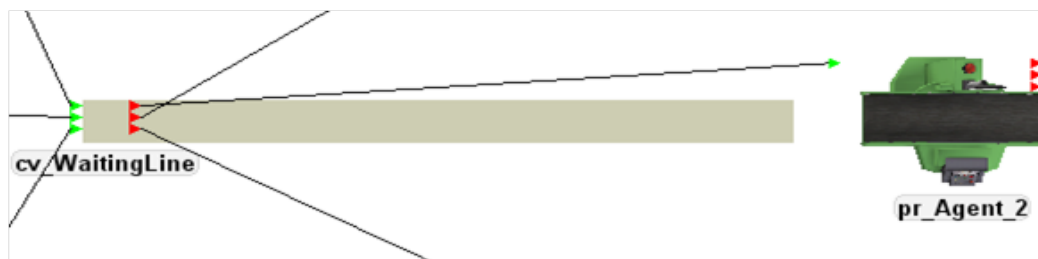


Figure A.7.7 Relationship between Flow triggers and port connections

of a passenger is Agent 2, since the processor representing Agent 2 is connected to Output Port 1 of the queue. This makes sense since the way the model is laid out, the queue is closest to (in front of) Agent 2. Certainly most people would choose the closest agent to where they are standing.

If Agent 2 is busy and Agents 1 and 3 are available, passengers would go to Agent 1; therefore, Agent 3 would only be used if the first two are busy. As a result, it would be expected that Agent 2 would be the most utilized and Agent 3 the least.

Setting the Processing Time

In the “More Lucky Air” exercise, the process time logic at the agents needs to be changed depending on the type of passenger. In the original model, each agent had its own process time since they only processed certain types of passengers. Now the agents can process any type of passenger. This is oftentimes referred to as parallel (or like) servers.

If the process time at the agent is determined at the source and stored in a flowitem label, the processor just needs to read the Service Time label for the current flowitem. An Expression can be typed into the Process Time trigger as shown in Figure A.7.8. This illustrates the use of a FlexSim command, `getabelnum()`. Notice that since the name of the label is written into a command the string name has to be enclosed in quotes (“”). Also be sure that the name matches exactly the label name on the item – otherwise a zero will be returned for a value.

FlexSim provides a large number of commands that make it easy to add to a model. As the name indicates, this command gets a numeric label value from an

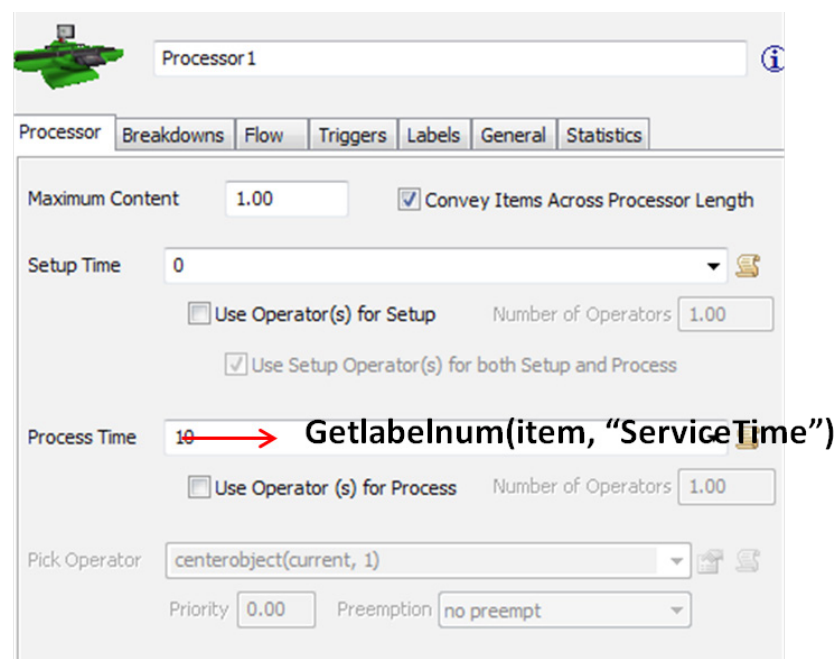


Figure A.7.8 Example processing time trigger logic using labels

object. The parameters of the command provide the specifics, in this case, to get the value stored on the flowitem (item) in the label named ServiceTime.

Other commands will be introduced throughout the book, but will be dealt with in more detail in a later portion of the book that discusses coding. The change to the agent needs to be made in all agent objects in the model.

Alternative for Service Time

It is common in modeling to have alternative ways to represent system behavior. There is usually no right or wrong way to model a system, although some ways may be better than others. Modelers, however, may not always agree on which approach is better. It all depends on what criteria is used to make the judgment, for example, clarity versus execution speed.

In order to illustrate this notion, consider an alternative way to model the “More Lucky Air” system. Instead of determining the processing time at the source and storing it in a label, the process time could be determined at the processor (agent) based on the incoming flowitem’s itemtype. Thus, the label ServiceTime is not needed, and it does not need to be set by the logic specified in the OnExit trigger of the source object. Instead, the logic in Figure A.7.9 is implemented at each agent processor.

The Process Time trigger uses the Cases By Value logic. While this coding construct is discussed in more detail later in the coding portion of the book, it is introduced here to illustrate other ways to implement logic in a model.

The way the logic works is that when the processor needs to determine a process time it first determines the itemtype of the current flowitem (via the `getitemtype()` command) and then executes the case statement corresponding to the current

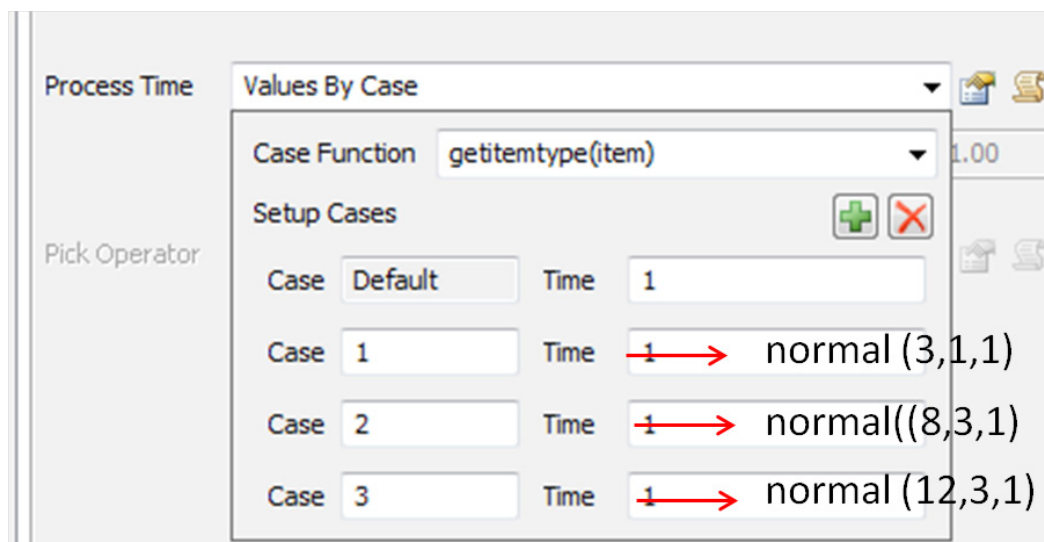


Figure A.7.9 Example processing time trigger logic using cases

itemtype value. For example, if the current flowitem has an itemtype value of 2 (a passenger with a paper ticket), the processing time would be 20 minutes, using the default logic. Of course this is not correct for “More Lucky Air”; therefore, the 20 is replaced by the `normal(8,3,1)` command. As a result, the processing time for itemtype 2 flowitems will be a sampled from the normal distribution with a mean of 8 minutes and a standard deviation of 3 minutes.

If a window opens at the bottom of the screen indicating an error has occurred, it usually means that the information was typed into the trigger template incorrectly (oftentimes due to incorrect spelling, inconsistent letter case, etc.). Check the accuracy of the trigger data. If no error is found, delete the trigger entry using the eraser icon, and try again.

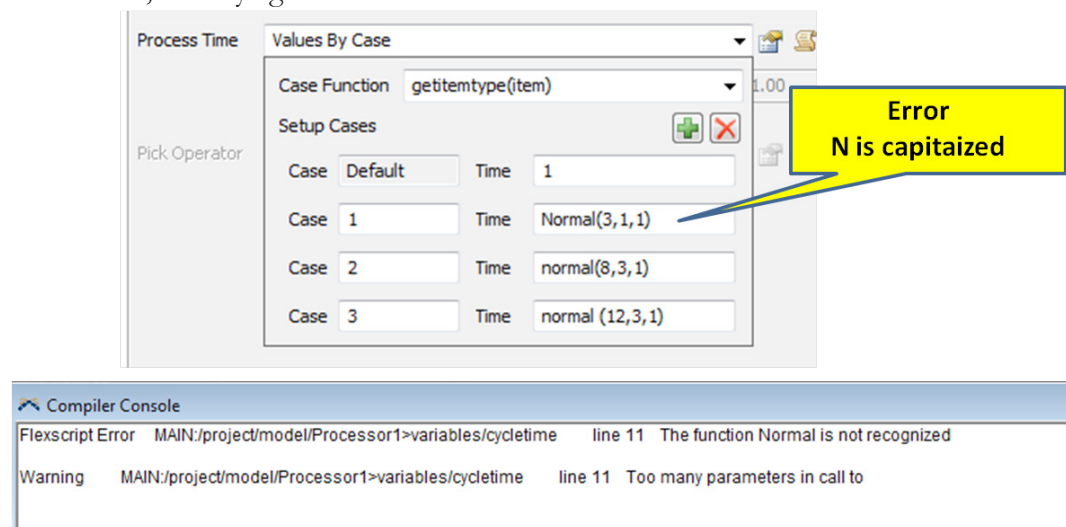


Figure A.7.10 Error in Process Time logic

Routing the exiting passengers

Finally, flows from the processors or agents to the sinks must be made properly since it is of interest to monitor the number of each type of passenger served.

The intent of the connections is to route flowitems out of the model through a sink that corresponds to their itemtype. All e-ticket passengers (itemtype = 1) should be sent to the sink named sk_ETicket; all passengers with paper tickets (itemtype = 2) should be sent to the sink sk_Paper, etc. Therefore, on the Flow tab on each processor (agent) the Send To Port trigger action should be set to By Expression. The default logic for this action is to route by itemtype. (Note that this implemented through the `getitemtype()` command.) As a result of the flow logic, itemtype 1 flowitems will exit a processor through Port 1, itemtype 2 though Port 2, etc.

To successfully complete the logic, it is important that sk_ETicket is connected to Output Port 1 from all processors, sk_Paper is connected to output Port 2 from all processors, etc. This is illustrated in Figure A.7.11.

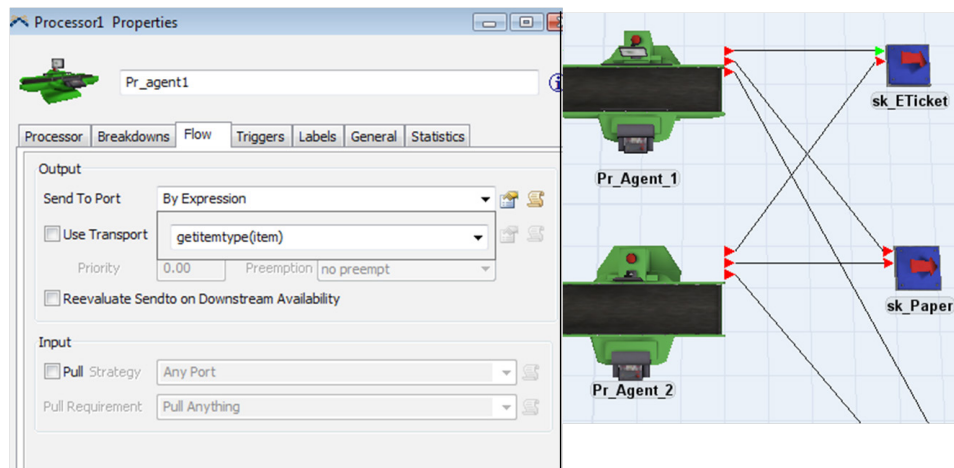


Figure A.7.11 Routing flowitems using Flow triggers and port connections

Output summary

Figure A.7.12 is a summary of some of the key outputs from a single run of the model for 168 hours. As in the case of the “Lucky Air” example in the last chapter, your results should be similar, but will not agree exactly with those in the table because it is likely that different random number streams were used.

Notice, for example, how much the average waiting time for the passengers is reduced when compared to the multi-queue operation in “Lucky Air.” Also, notice that the highest utilization is for Agent 2 and the lowest is for Agent 3; this is as expected based on the discussion above.

	Waiting line	E-ticket	Paper ticket	Purchase ticket
Number of customers served		2085	978	668
Throughput (customers per hour)		12.4	5.8	4.0
Agent utilization		73.3	81.9	64.9
Average lengths of waiting line	1.2			
Maximum length of waiting line	14			
Average (max) waiting time	2.96 (28.45)			

Figure A.7.12 Output for exercise “More Lucky Air”

Section 4 Exercise 7-2 Even More Lucky Air

One use of the global table is associated with an empirical distribution in *FlexSim*. Any of these distributions (dempirical, empirical, cempirical) require a global table where probability percentages must be entered in column 1 starting with row 1 of

the table, and their associated values, such as itemtype, are entered in column 2. The table may have as many rows as needed to define as many values as desired. The percentages are entered as numbers between 0 and 100, and must add up to a total of 100 percent.

- Use a single source that feeds both the general customer line and the high roller line. Based on the previous analysis, assume the time between arrivals is exponentially distributed with a mean of 2.7 minutes.
- Now that there is only a single source, the logic must change for how a flowitem's itemtype is set, as well as determining the service time.
- When a flowitem is created, its itemtype is set randomly based on the percentages calculated earlier and as stored in the global table created in the last section. That is, each flowitem has a 54.5% chance of being an e-ticket passenger, 27.3% chance of being a passenger with a paper ticket, etc. As shown in Figure A.7.13, this is implemented in the OnCreation trigger of the single source object. The Set Itemtype picklist option is edited to use the dempirical command rather than the default duniform(1,3) command. When executed, the dempirical command randomly samples from the empirical distribution specified in the named global table. The second parameter is the random number stream. The command will return an itemtype of 1, 2, or 3.

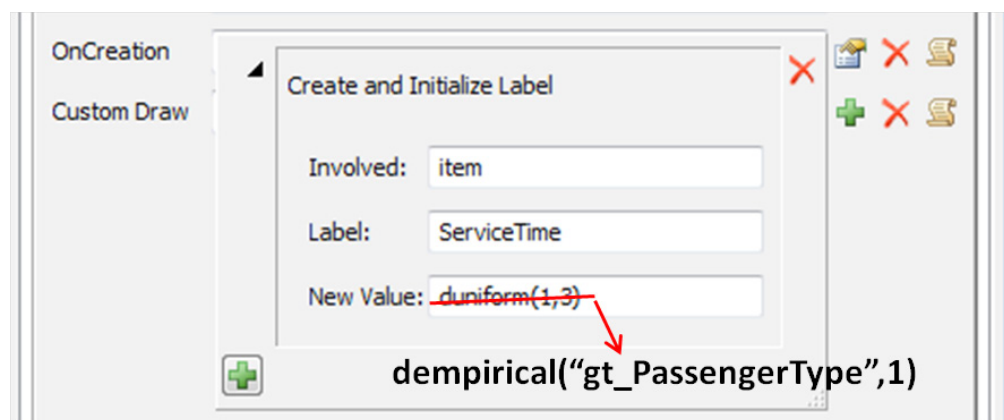


Figure A.7.13 Using a global table and the dempirical command to set itemtype

- The flowitem's color is set in the same way as the previous exercise.
- The processing time by the agent will not be set at the source, but will be determined at the processor using the Case By Value option in the Processing Time trigger.
- Route the passengers from the source to the appropriate queue using the Discrete Empirical Distribution picklist option that is available in the Send To Port trigger on the Flow tab. In order to do this, another global table needs to be created that contains the percentage of each priority level

(regular and high roller) and the corresponding port number where the flowitem should be routed.

- Make sure the connections between the queues and the processors (agents) are made so that agents select high rollers first.

A single run of 168 hours of the model should result in output values similar to those in Figure A.7.14. As discussed before, the values may differ somewhat due to the use of different random number streams. Note the significantly less wait time for the high rollers as compared to the regular customers.

	Waiting line reg.	Waiting line HR	E-ticket	Paper ticket	Purchase ticket
Number of customers served			2029	1001	670
Throughput (customers per hour)			12.1	6.0	4.0
Agent utilization			73.9	80.7	63.9
Average length of waiting line	1.19	0.04			
Max length of waiting line	14	3			
Average (max) waiting time	3.59 (36.76)	1.10 (11.74)			

Figure A.7.14 Example Output for Exercise 7-2 “Even More Lucky Air”

Appendix for Chapter 8

This Appendix provides further information on Exercise 8-1, Hampton International. It describes working with task executors and time tables in *FlexSim*. It also provides additional information on Exercise 8-2, Steve's Stone Cutting, and Exercise 8-3, The Crafty Framer.

Section 1 Exercise 8-1 Hampton International

Application Notes: (this is only one of many ways to build the simulation)

- Use the object flow diagram provided in Figure A.8.1..
- Use a pallet flowitem to represent the transporters.
- Use a source and select the Arrival Sequence option in the Arrival Style picklist to load the return conveyor with transports at the start of the simulation. Note: the transports may look like one object when they start out. Be sure all transports are at the loading station.
- Use a combiner to put the bags on the transporter with the return conveyor as the first input.
- You need use only one processor for the inspection station, with the number of inspectors listed as the capacity.

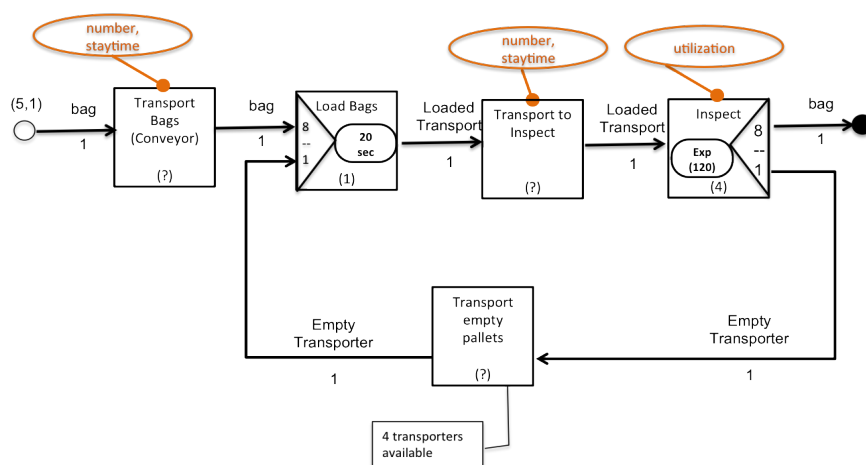


Figure A.8.1 Object Flow Diagram for Hampton International

- Use a separator to separate the bags from the transporter, with the first output going to the return conveyor.
- Color the bags to make them more visible

Some statistics for the base case operation	8 hours
Bags to the transfer area	2119
Percent of time bags backed up outside of transfer	11.3%
Rate of bags delivered to the distribution system	4.2 per min
Average wait time for bags to be loaded on transport	19.5 min

An example model layout is shown in Figure A.8.2

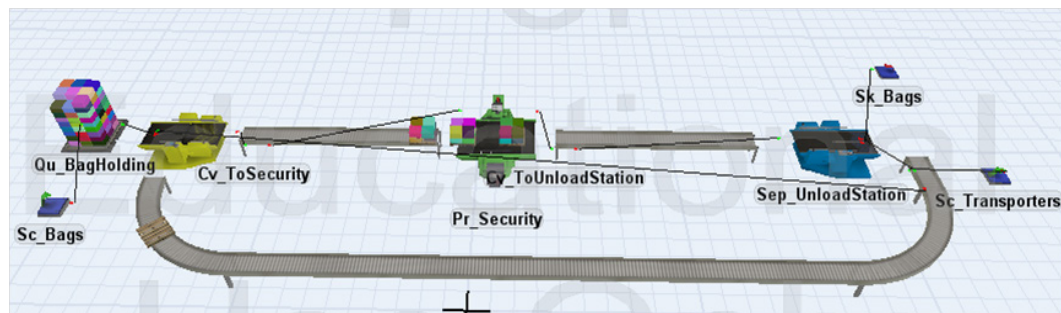


Figure A.8.2 Hampton International Simulation

Section 2 Connecting and calling task executers

As shown in Figure A.8.3, task executers are connected to objects through a center port connection. A center port connection is made by holding down the S key, selecting the calling object (the one requesting the task executor, for example, a processor that requires an operator for setup), and dragging the mouse from the object to the task executor.

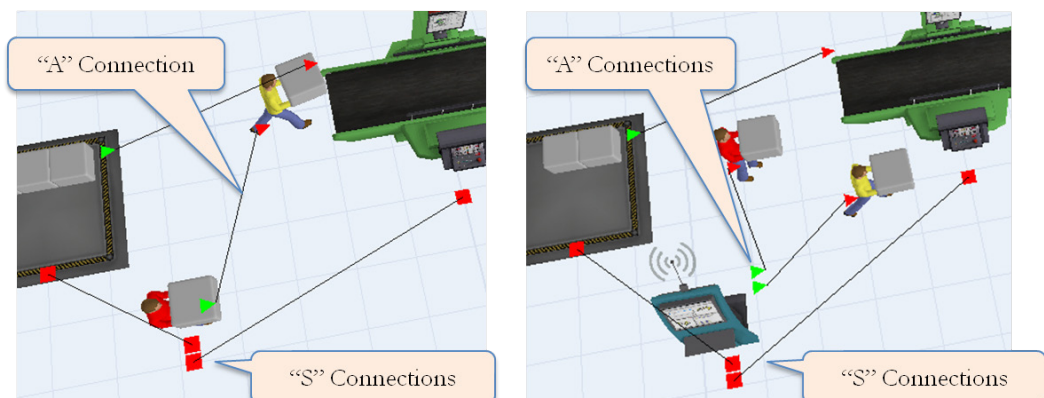


Figure A.8.3 Task executor connections

In the case on the left side of Figure A.8.3, a task executor shares tasks with another task executor; the task executors are connected through a standard “A” connection. The queue in the figure uses an operator to move flowitems from the queue to the processor and the processor uses an operator either for setup or processing, or both. Either operator can do either task. The task executor connected to the queue and processor objects is like a working supervisor - it does tasks and, when busy, passes those tasks along to the other operator.

Another approach is shown on the right side of Figure A.8.3. In this case, a *Dispatcher* object is used. The dispatcher does not do any of the tasks, it only distributes them to task executors connected to it via “A” connections. While any task executor can share tasks, the dispatcher is specifically designed to control the sharing of tasks among many task executors.

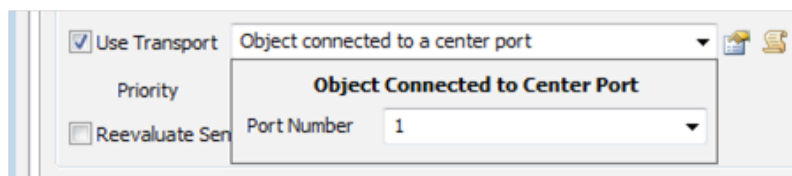


Figure A.8.4 Assigning a task executor

Task executors are assigned, or called, by an object through a logic statement. As shown in Figure A.8.4, task executors are assigned for a transport task by selecting the Flow tab of the object and checking the Use Transport box. A priority level may also be set. *Preempt* means that this task can be preempted or interrupted by higher ranking tasks.

The default drop-down choice is to use the task executor connected to the first center port. Note that the choice of port number, or other logic, can be changed. A task executor is assigned for a setup or process task in a similar way through the Use Operator check box on the main window of the object.

Network Nodes

Network node objects are used to provide specific paths for task executors to follow as they move around the model surface. The primary interface tab for a network node is shown in Figure A.8.5.

Establishing a network for task executors consists of three steps:

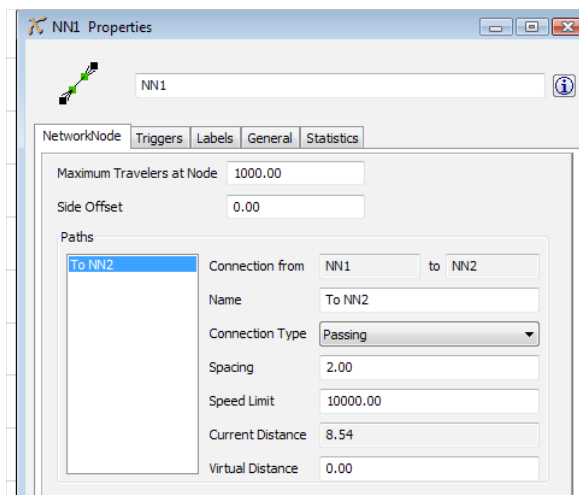


Figure A.8.5 Network node functions

1. Configuring the path

Paths are created by laying out the network nodes on the simulation surface and connecting nodes to form paths.

- Use the A-drag connection to connect the nodes.
- Each node can have several paths to other nodes.
- Each path represents two single-direction segments.
- Each path between each pair of nodes can be independently configured from the node window.
 - *Passing*: Task executors can pass each other on the path; this is indicated by a green dot on the path.
 - *No Passing*: Task executors can travel single file only; this is indicated by a yellow dot on the path.
 - *No Connect*: Task executors are allowed no travel in the particular direction; indicated by a red dot on the path.
 - *Speed Limit*: This configuration restricts the speed set on the task executor.
 - *Virtual Distance*: This setting overrides the actual distance determined by the grid on the model surface, if the value is greater than 0. It is used to specify a distance independent of the simulation layout.
- The path's visual appearance can be configured by right clicking on the colored dots on the path..

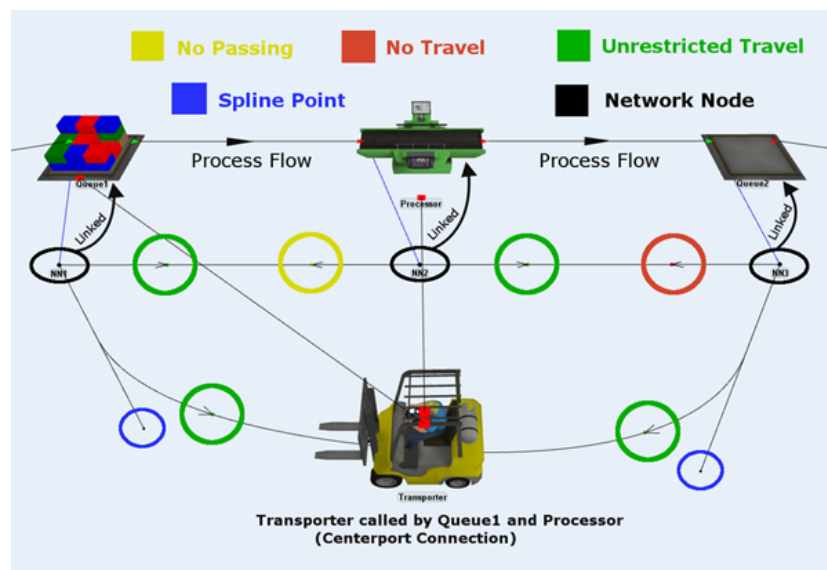


Figure A.8.6 Behavior of connections on a Path Network

- A curved path creates two “handles” that can change path’s shape.

The three travel options (passing, no passing, and no connect) are illustrated in Figure A.8.6. In the first case, between Nodes 1 and 2 (NN1 and NN2), the green and yellow dots on the path indicate task executors can move in either direction, but can only pass moving to the right (from Node 1 to 2).

In the second case, between Nodes 2 and 3 (NN2 and NN3), the green and red dots on the network path indicate task executors can move only to the right (from Node 2 to 3) and not from Node 3 to 2; faster task executors can pass slower ones going to the right.

In the third case, on the curved path between Nodes 1 and 3 (NN1 and NN3), the two green dots on the path indicate task executors can move in either direction, from Nodes 1 to 3 and from 3 to 1; faster task executors can pass slower ones in either direction.

Figure A.8.6 also shows the connections from the network nodes to their calling objects and to the task executor.

2. Connect network nodes to involved objects

- Nodes are gateways to and from involved objects (objects that interact with task executors on a path network, such as a queue calling a task executor to pick up a flowitem or a processor receiving a flowitem from the task executor).
- Use the ‘A’ drag connection to connect nodes to objects; when a connection is made, a blue line appears.
- Objects may be connected to multiple nodes.
- A node may be connected to multiple objects.

Note in Figure A.8.6 Queue1 is connected to NN1, Processor1 is connected to NN2, and Queue2 is connected to NN3.

3. Connecting task executors to network nodes

- To move on a network, task executors must be connected to a node.
- Use the ‘A’ drag connection to connect task executors to nodes; when the connection is made, a red line appears.
- When the model is reset, executors move to the node they are connected to.
- Multiple task executors can be attached to a node.

Task executors still have to be assigned to an object with a center connection, as discussed earlier. In the case of Figure A.8.6, the transporter is connected to (and

called by) Queue1 and Processor1. The objects also must still be connected together to indicate the flow path. In the example in Figure A.8.6, the process flow for the flowitems is from Queue1 to Processor1 to Queue2. The flowitems move between those fixed resources through the use of the transporter that travels on the network path defined by NN1, NN2, and NN3. For more detailed information, refer to the *FlexSim* User Manual.

Section 3 Configuring timetables

There are two independent ways to set up the table. The two methods are not synchronized and only one should be used for any one table. For simple occurrences, manually filling out the timetable on the main window is best.

Consider an example of a small job shop that operates 12 hours a day. At the end of the day everyone leaves, and then on the next day they pick up where they left off. There are three break times when no orders are accepted and all work stops.

- A 30 minute break after 3 hours (180 minutes)
- A one hour break after 5 hours (300 minutes)
- A 30 minute break after 8 hours (480 minutes)

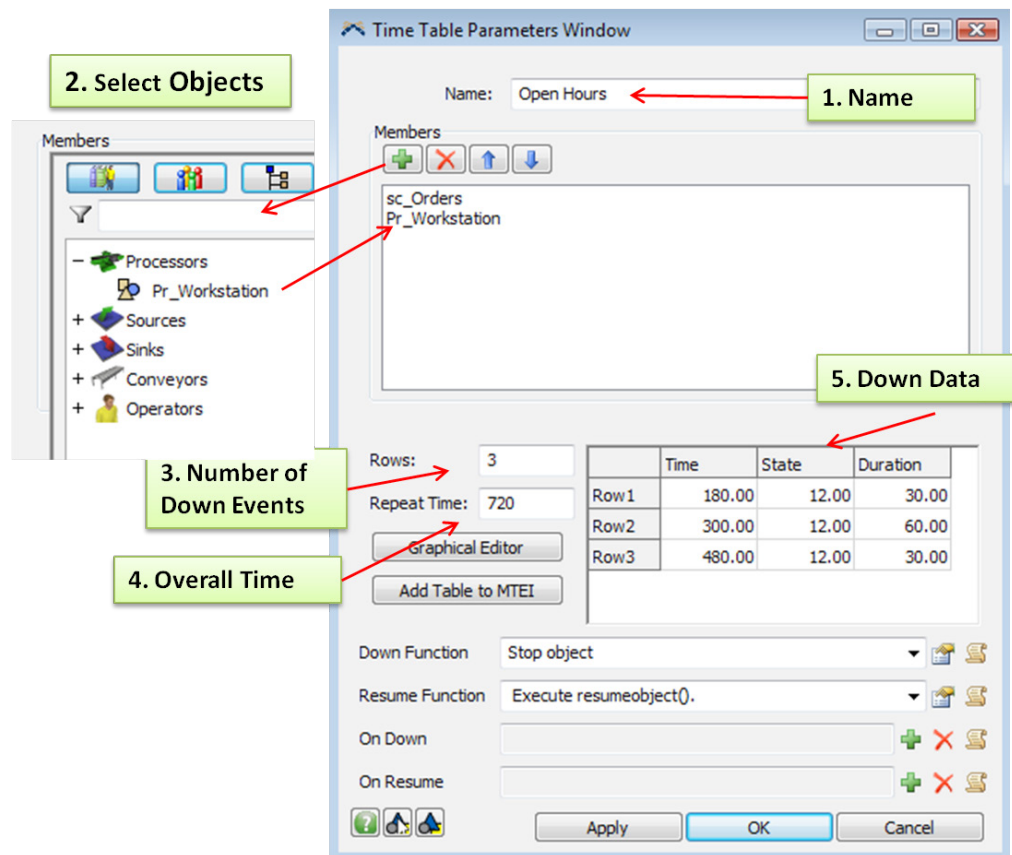


Figure A.8.7 Creating a Time Table manually

As illustrated in Figure A.8.7, the following steps are involved to manually set up the table:

1. Create a name for the table.
2. Select the objects to stop (in this case, the order source and machine).
3. Determine the number of downtime events (in this case 3).
4. Select the overall time for the operation (12 hours or 720 minutes).
5. Pick a down state from drop-down list.
6. Fill out the table cells.

For a more complex timetable, the graphical editor is used.

Consider the above example, but establish a table where the operation is open from 8 a.m. to 8 p.m. four days a week and a half day on Friday.

The following steps are involved to set up the table with the graphical editor, as shown in Figure A.8.8:

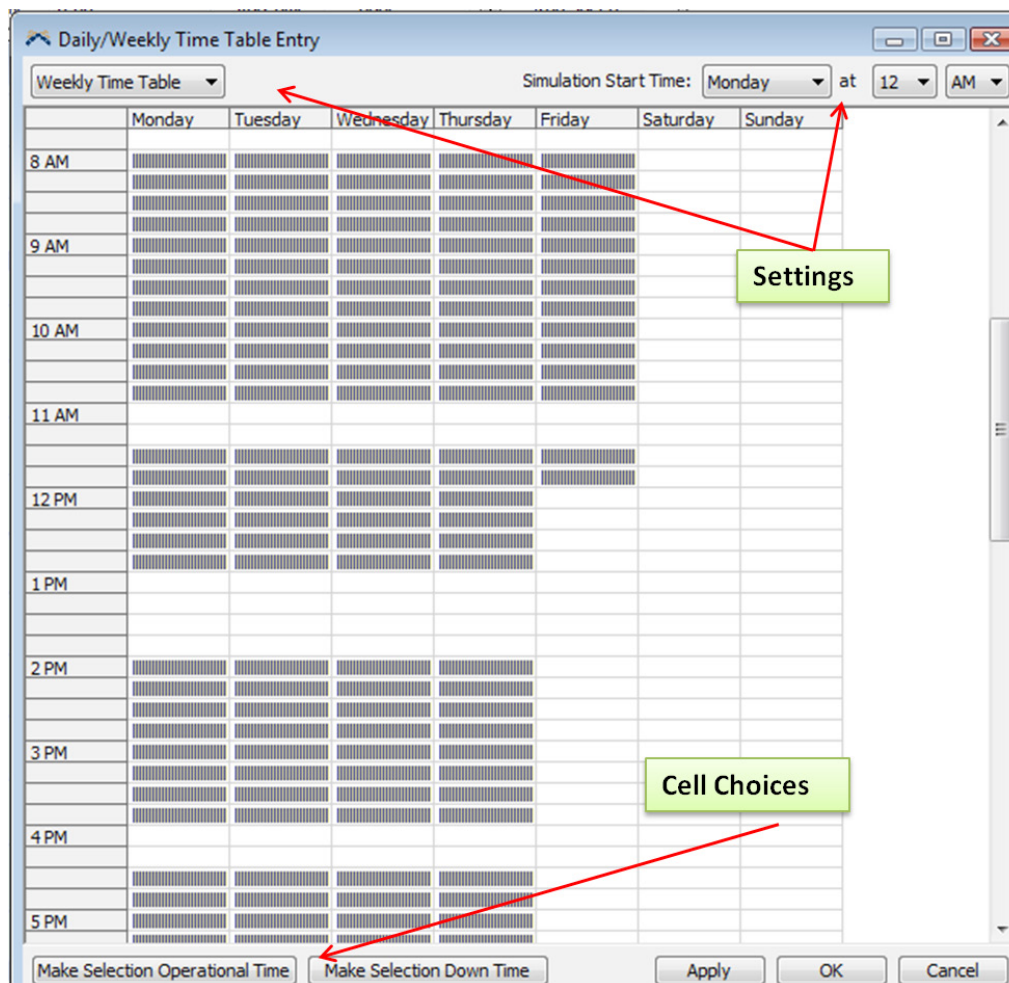


Figure A.8.8 Using the Graphical Editor to set up a timetable

1. Create a table with a new name.
2. Select the objects to stop (the same as before—order source and machine).
3. Select the Graphical Editor button to open the interface shown in Figure A.8.8.
4. Select the settings at the top of the window:
 - Default start time is Monday morning at midnight.
5. Select the cells by clicking and dragging and then choosing the appropriate option at the bottom of the page for that range (operational time or downtime).
 - The operational state is filled in; downtime is empty.
 - The default operational time is 8 a.m. to 4 p.m. Monday through Friday.
6. Click Apply and OK.
7. The table is filled out for the entire week starting at midnight Monday morning.

Section 4 Exercise 8-2 Steve's Stone Cutting

Application Notes: (this is only one of many ways to build the simulation) An example model layout is shown in Figure A.8.9.

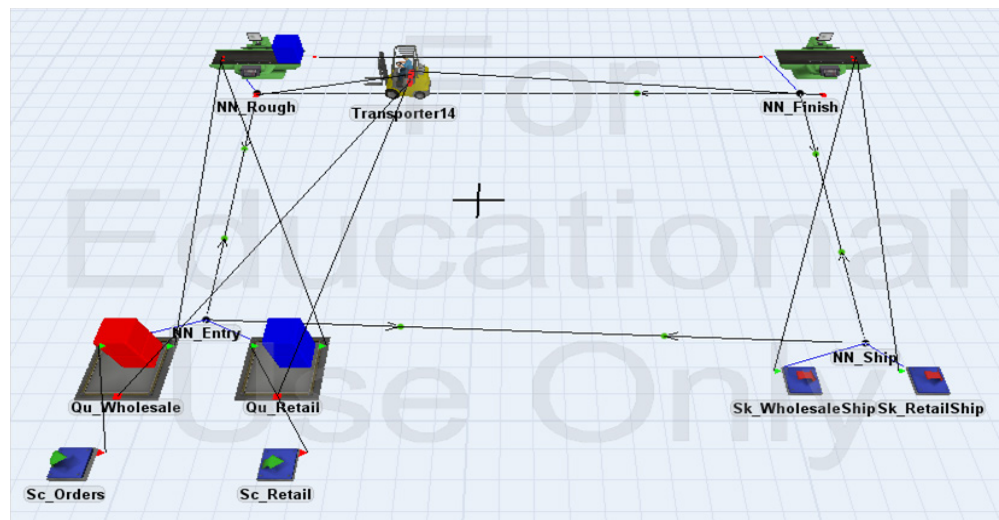


Figure A.8.9 Steve's Stone Cutting model

- Be sure to order the input ports to the rough cutting machine so that wholesale projects get priority. Later, you can get rid of any priority by disconnecting the retail source and connecting it to the wholesale order queue.

- Utilize the virtual distance function on the network nodes to establish the distance. Be sure to get the right distances on the correct node connections.
- Make an additional network node connection to add alternative paths.
- Add another fork truck by making an A connection from the first to the second. Be sure that all the attributes on the second transporter are correct. Hint: Duplicating (copying and pasting) the first transporter will help ensure all properties of the first transporter are included in the second.

Section 5 Exercise 8-3 The Crafty Framer

Application Notes: (this is only one of many ways to build the simulation) An example model layout is shown in Figure A.8.10.

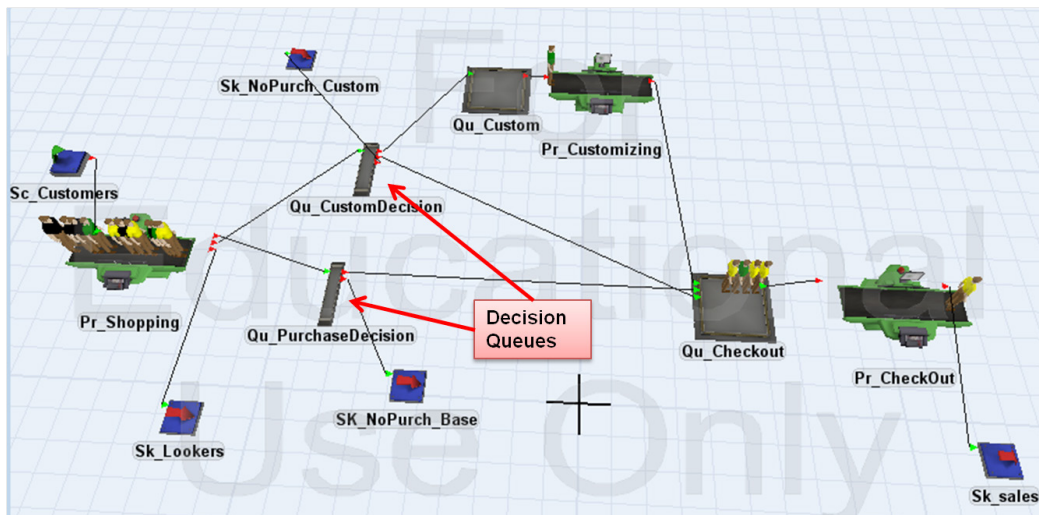


Figure A.8.10 Frame Shop model

- Use the demperical command to establish item types for shoppers.
- Use the Maximum Content for the number of shoppers. Send customers out of the shopping processor to the right spot by item type.
- Set item types in source trigger.
- Use queues before the custom and checkout processors.
- Use an additional queue in front of each of the previous queues to direct people elsewhere if the queues in front of them are full.

Stop the customer source by using a time table to represent the store closing and clean out time. Start the time table when the store opens. The store is then closed after 10 hours for 2 hours. The cycle repeats. See Figure A.8.11.

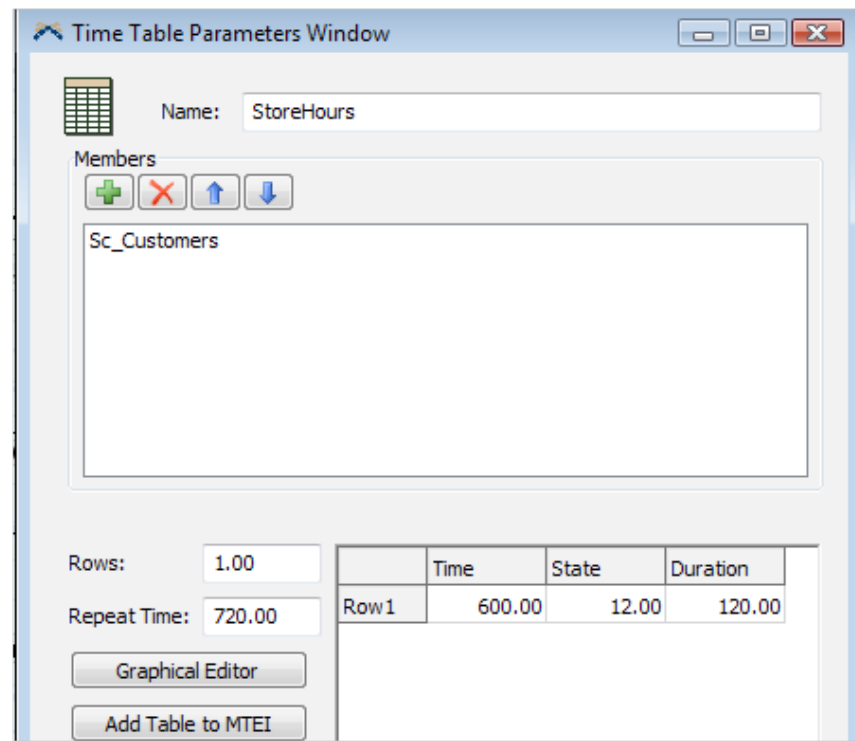


Figure A.8.11 Time Table for Frame Shop

Typical results for a run of 5 days is shown in the following table. The results show the total number of customers during the time period. They are broken down by type and their actions.

Total Customers			2711
Base Customers		973	
Frame purchase	790		
Left with no purchase	183		
Custom Customers		678	
Custom frame	508		
Frame only	126		
Left with no purchase	44		
Lookers		1424	

Appendix for Chapter 9

This Appendix provides instructions on using *ExpertFit* and information on some of the discrete and continuous statistical distributions available in *FlexSim*. It also provides tables for the standard normal distribution and Student's t distribution.

Section 1 Notes for Using *ExpertFit*

ExpertFit not only provides the ability to analyze raw data, but it also allows the user to view statistical distributions and estimate distributions when only general observations are available. The application provides two modes of operation when determining what distribution best fits a data set: Standard Mode, which fits most cases, and an Advanced Mode, which contains additional features for an advanced user. There are also two levels of precision for the data analysis. The *ExpertFit* application contains an extensive Help file as well as tutorials.

The first step in using the application to fit raw data involves collecting a data set to be used for the analysis. *ExpertFit* recommends the following when collecting data for analysis.

- If, at all possible, *collect at least 100 observations* on the random phenomenon of interest, with 200 observations providing more ability to discriminate between two distributions. In general, the benefit from increasing the sample size from 200 to 300 will be less than that provided by increasing the sample size from 100 to 200.
- If collected observations are from a continuous random variable (e.g., a service time), then the data values should have enough resolution so that the sample will have a large number of distinct values. Otherwise, it will be difficult, in general, to find a continuous distribution that provides a good representation.
- If the available data values are integers, then you may want to convert them to real numbers. *ExpertFit* contains many more continuous distributions than discrete distributions.
- You should understand the process that produced the data, rather than treating the observations as just abstract numbers. For example, suppose your data set contains a few extremely large observations; these are called outliers. If you don't understand the problem context, then it will be

difficult to know whether these large observations are really legitimate or, perhaps, the result of measuring or recording errors.

- If you have collected times of arrival of “customers,” then these can be converted to interarrival times using the *ExpertFit* transformation, DIFF.

The following steps are used to obtain a target distribution with the *ExpertFit*.

1. Obtain a data set using the Data tab; see Section 1.2 of the *ExpertFit* manual for a discussion of the type and amount of data required.
2. View the resulting Data-Summary Table (at the Data tab); it provides information on the shape and range of the true density function.
3. Construct a histogram of the data (used in Step 5) using the Data tab; see the “Constructing a Histogram from Your Data” tutorial in the online help.
4. Select the distribution that is the best representation for your data using the Automated Fitting option at the Models tab.
5. Confirm using the Comparisons tab that the best distribution as determined by *ExpertFit* is, in fact, satisfactory in an absolute sense. See Section 2.1 of the *ExpertFit* manual for recommendations.
6. If you are doing simulation modeling, then represent either the best-fitting distribution (if good in an absolute sense) or an empirical distribution based on your data (if the best distribution is not satisfactory) in *FlexSim* using the Applications tab.

Four examples of the use of the Data Analysis module are given in Section 2.2 of the *ExpertFit Version 7 User's Guide*.

Section 2 Statistical distributions in *FlexSim*

A complete set of statistical commands with examples can be found in the *FlexSim* Help section. This appendix serves to summarize a few of the more common distribution options.

Note: The abbreviations num and str stand for number and string, which means that the corresponding parameter should be a numerical value or a text string. Text is always placed between double quotes, the symbols “ and ”. The last parameter “num stream” refers to the number of the stream for the random number generator. As discussed later, for single scenario analyses, its value can remain 1 or can even be omitted as a parameter. However, when using a model for comparisons, it is best to

assign each source of randomness (interarrival time, service time, quality indicator, etc.) a unique stream number.

Discrete probability distributions

bernoulli(num percent-prob, num succeed-value, num fail-value, num stream)

A discrete random variable takes on the value succeed-value with probability percent-prob and fail-value with probability (100-percent-prob).

Example: `bernoulli(90, 0, 1, 2)` Based on an inspection process, this generates either a 0 or 1 representing a good or bad part, respectively. There is a 90% chance the tested product is good (returns a value of 0); similarly, there is a 10% chance the part is bad (returns a value of 1). It uses random number stream 2. To specify more than two possible values, such as good, bad, rework, see the commands `empirical` or `dempirical`.

binomial(num trials n, num percent-prob, num stream)

A discrete random variable of the count of the number of successes in n trials where each trial has a success rate percent-prob.

Example: `binomial(20, 16.67, 2)` This simulates 20 trials with a die where we count the number of times a 1 occurs (probability of a 1 on each trial is one-sixth or 16.67%) using random number stream 2.

duniform(num min, num max, num stream)

Generates equally likely discrete values between min and max.

Example: `duniform(1, 6, 3)` simulates a die in that every value from 1 to 6 has 16.66 percent probability of returning 1, 2, ..., 6, using random number stream 3. See also the continuous distribution command `uniform`.

dempirical(str tablename, num stream)

This is a discrete empirical distribution function. It randomly returns a discrete value that is defined in Column 2 of a global table. It is expected that Column 1 of the table contains the percent occurrences of the values and Column 2 contains the values. The percents in Column 1 must sum to 100.

Example: `int output = dempirical("SendToTable", 4);`

This sets the local variable `output` to a random value based on the distribution specified in the global table named `SendToTable` using random number stream 4.

Continuous probability distributions

empirical(str tablename, num stream)

This is a continuous empirical distribution function. It randomly returns a continuous value that is defined in Column 2 of a global table. It is expected that Column 1 of the table contains the percent occurrences of the values and Column 2 contains the values. The percents in Column 1 must sum to 100. The values in Column 2 must be in ascending order because this function estimates values between the specified values continuously.

Example: `double ptime = empirical("ProcessTimes", 1);`

This sets the local variable `ptime` to a random value based on the distribution specified in the global table named `ProcessTimes` using random number stream 1.

exponential(num location, num scale, num stream)

The location parameter, or lower bound of possible outcome values, is typically 0. The scale represents the parameter of the average service or inter-arrival time.

Example: `exponential(0, 120, 2)` This simulates a process with a mean time between arrivals of 120 (seconds), where the outcome will be in the range of 0 to infinity, using random number stream 2.

erlang(num location, num scale, num shape, num stream)

When the location parameter, or lower bound of possible outcome values, is 0 then the common notation is m-Erlang(β) or shape-Erlang(scale). The erlang distribution is the sum of m identically distributed exponential distributions with mean β . Thus, the exponential (location, scale) = `erlang(location, scale, 1)`. The mean is `location + scale*shape`, mode = `location*scale*(shape-1)`, and variance = `scale^2*shape`.

Example: `erlang(0, 20, 4, 3)` This simulates an assembly process composed of four steps, each taking 20 seconds, using random number stream 3.

normal(num mean, num stddev, num stream) (see Figure A.9.1)

This samples from the unbounded continuous distribution where the first parameter is the mean of the distribution and the second is the standard deviation.

Example: `normal(100, 10, 4)` simulates a process with a mean of 100 and a standard deviation of 10 using random number stream 4. Many results will be in the neighborhood of 100.

triangular(num min, num max, num mode, num stream)

This command returns a value between `num min` and `num max`; `num mode` is the value where the probability density function (pdf) is maximized. Warning: the mode is different from mean, but in the case of the triangular, $\text{mean} = (\text{min} + \text{mode} + \text{max}) / 3$. So, if you know the mean you can calculate the mode.

Example: `triangular(3.5, 10, 5, 5)` Time to complete a task is triangularly distributed with the shortest process time being 3.5 (min.), the longest process time being 10 (min.), and the most likely process time being 5 (min.). Random number stream 5 is used to sample the values.

`uniform(num min, num max, num stream)`

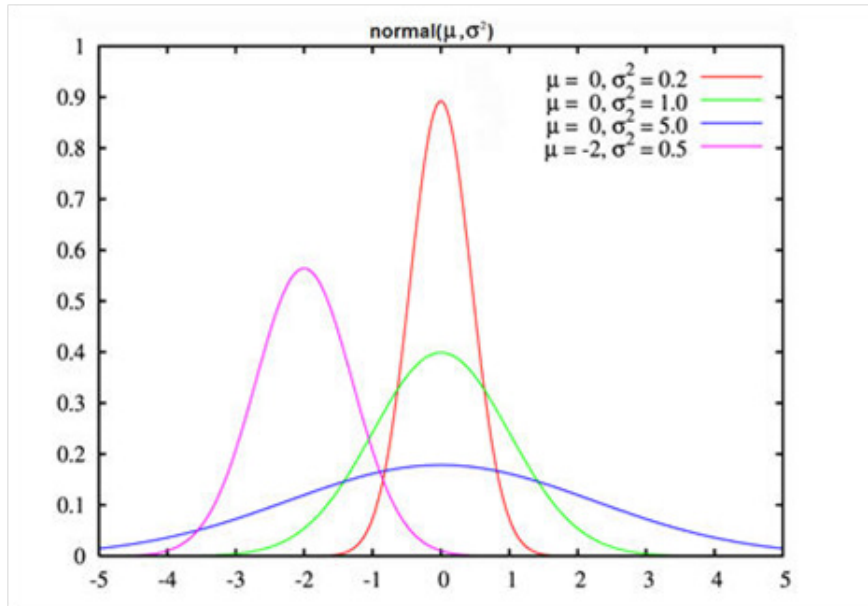


Figure A.9.1 Example normal probability density functions

This generates a continuous value that is equally likely between a specified lower and upper bound of the interval given by the parameters min and max, respectively.

Example: `uniform(1, 7, 6)` A travel time is uniformly distributed between 1 and 7 (min.). Random number stream 6 is used. See also the command `duniform` for cases when the sampled values need to be discrete.

`beta(num min, num max, num shape1, num shape2, num stream)`

This continuous distribution is bounded between min and max but can take on a variety of shapes based on the two non-negative shape parameters. The mean is

$$\text{mean} = \text{min} + (\text{max} - \text{min}) \frac{\text{shape1}}{(\text{shape1} + \text{shape2})}.$$

Note:

- `beta(min, max, 1, 1) = uniform(min, max)`
- `beta(min, max, 1, 2) = triangular(min, max, min)`, a left triangle.
- `beta(min, max, 2, 1) = triangular(min, max, mode)`, a right triangle.

A variety of beta distributions are shown in Figure A.9.2.

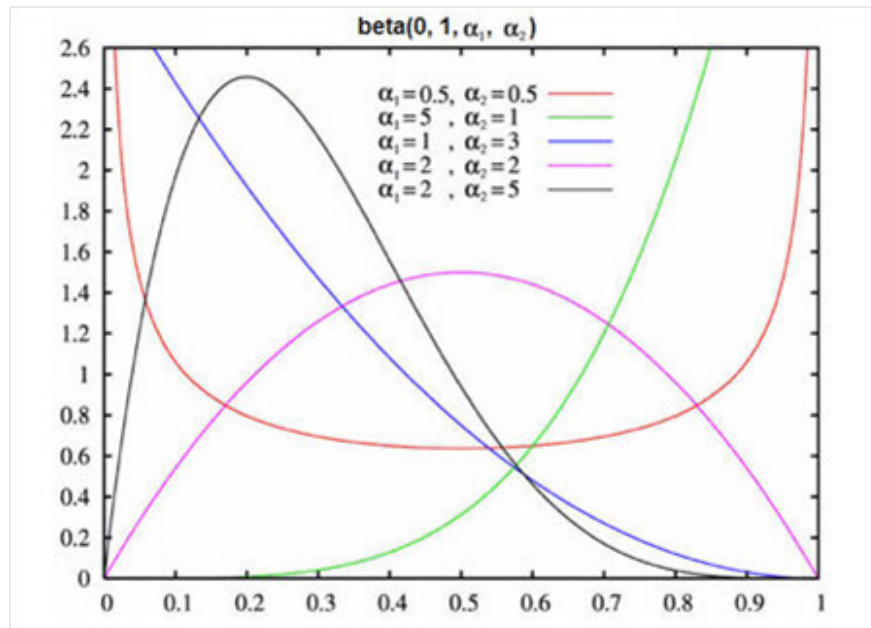


Figure A.9.2 Examples of beta probability density functions

gamma(num location, num scale, num shape, num stream)

This continuous distribution can take on a variety of shapes, as shown in Figure A.9.3., based on its parameters. The gamma (location, scale, 1) = exponential(location, shape) and the gamma (location, β , m) = erlang(location, β , m).

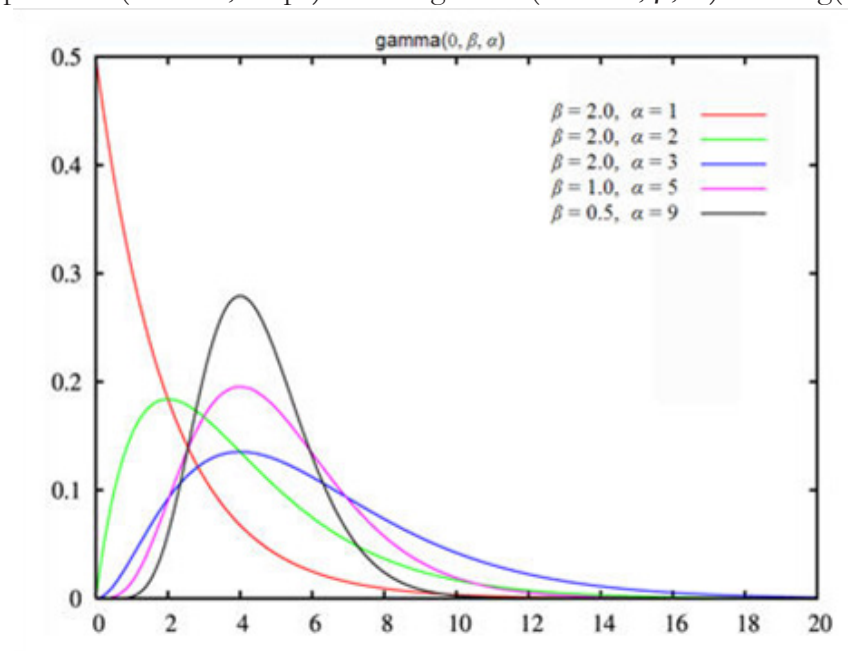


Figure A.9.3 Examples of gamma probability density functions

weibull(num location, num scale, num shape, num stream)

This non-negative continuous distribution can take on a variety of shapes, as shown in Figure A.9.4, based on its parameters. The `weibull(location, scale, 1)` = `exponential(location, shape)`.

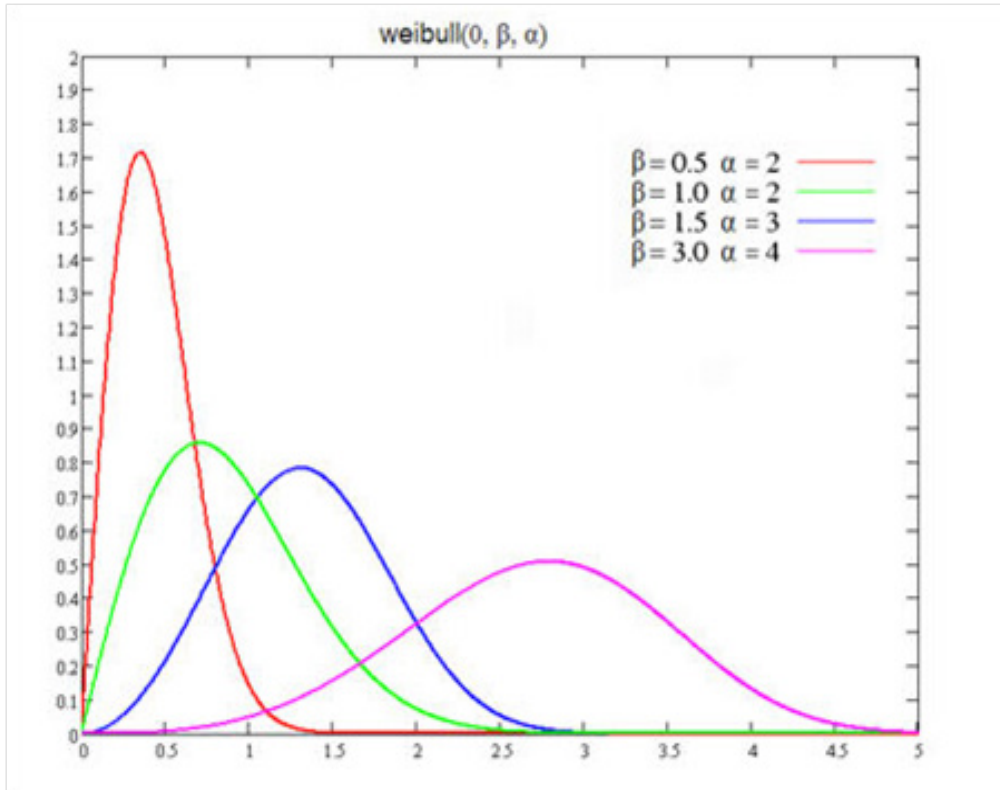


Figure A.9.4 Examples of Weibull probability density functions

Section 3 Tables of normal and t distributions

The following tables are public domain and are produced by an APL program written by the author, William Knight; accessed at:

<http://www.math.unb.ca/~knight/utility/NormTble.htm> and

<http://www.math.unb.ca/~knight/utility/t-table.htm>

Tables of the Normal Distribution



Probability Content from $-\infty$ to Z

Z	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

PERCENTAGE POINTS OF THE T DISTRIBUTION									
Tail Probabilities									
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
-----+-----									
D	1	3.078	6.314	12.71	31.82	63.66	318.3	637	1
E	2	1.886	2.920	4.303	6.965	9.925	22.330	31.6	2
G	3	1.638	2.353	3.182	4.541	5.841	10.210	12.92	3
R	4	1.533	2.132	2.776	3.747	4.604	7.173	8.610	4
E	5	1.476	2.015	2.571	3.365	4.032	5.893	6.869	5
E	6	1.440	1.943	2.447	3.143	3.707	5.208	5.959	6
S	7	1.415	1.895	2.365	2.998	3.499	4.785	5.408	7
	8	1.397	1.860	2.306	2.896	3.355	4.501	5.041	8
O	9	1.383	1.833	2.262	2.821	3.250	4.297	4.781	9
F	10	1.372	1.812	2.228	2.764	3.169	4.144	4.587	10
	11	1.363	1.796	2.201	2.718	3.106	4.025	4.437	11
F	12	1.356	1.782	2.179	2.681	3.055	3.930	4.318	12
R	13	1.350	1.771	2.160	2.650	3.012	3.852	4.221	13
E	14	1.345	1.761	2.145	2.624	2.977	3.787	4.140	14
E	15	1.341	1.753	2.131	2.602	2.947	3.733	4.073	15
D	16	1.337	1.746	2.120	2.583	2.921	3.686	4.015	16
O	17	1.333	1.740	2.110	2.567	2.898	3.646	3.965	17
M	18	1.330	1.734	2.101	2.552	2.878	3.610	3.922	18
	19	1.328	1.729	2.093	2.539	2.861	3.579	3.883	19
	20	1.325	1.725	2.086	2.528	2.845	3.552	3.850	20
	21	1.323	1.721	2.080	2.518	2.831	3.527	3.819	21
	22	1.321	1.717	2.074	2.508	2.819	3.505	3.792	22
	23	1.319	1.714	2.069	2.500	2.807	3.485	3.768	23
	24	1.318	1.711	2.064	2.492	2.797	3.467	3.745	24
	25	1.316	1.708	2.060	2.485	2.787	3.450	3.725	25
	26	1.315	1.706	2.056	2.479	2.779	3.435	3.707	26
	27	1.314	1.703	2.052	2.473	2.771	3.421	3.690	27
	28	1.313	1.701	2.048	2.467	2.763	3.408	3.674	28
	29	1.311	1.699	2.045	2.462	2.756	3.396	3.659	29
	30	1.310	1.697	2.042	2.457	2.750	3.385	3.646	30
	32	1.309	1.694	2.037	2.449	2.738	3.365	3.622	32
	34	1.307	1.691	2.032	2.441	2.728	3.348	3.601	34
	36	1.306	1.688	2.028	2.434	2.719	3.333	3.582	36
	38	1.304	1.686	2.024	2.429	2.712	3.319	3.566	38
	40	1.303	1.684	2.021	2.423	2.704	3.307	3.551	40
	42	1.302	1.682	2.018	2.418	2.698	3.296	3.538	42
	44	1.301	1.680	2.015	2.414	2.692	3.286	3.526	44
	46	1.300	1.679	2.013	2.410	2.687	3.277	3.515	46
	48	1.299	1.677	2.011	2.407	2.682	3.269	3.505	48
	50	1.299	1.676	2.009	2.403	2.678	3.261	3.496	50
	55	1.297	1.673	2.004	2.396	2.668	3.245	3.476	55
	60	1.296	1.671	2.000	2.390	2.660	3.232	3.460	60
	65	1.295	1.669	1.997	2.385	2.654	3.220	3.447	65
	70	1.294	1.667	1.994	2.381	2.648	3.211	3.435	70
	80	1.292	1.664	1.990	2.374	2.639	3.195	3.416	80
	100	1.290	1.660	1.984	2.364	2.626	3.174	3.390	100
	150	1.287	1.655	1.976	2.351	2.609	3.145	3.357	150
	200	1.286	1.653	1.972	2.345	2.601	3.131	3.340	200
-----+-----									
Two Tails		0.20	0.10	0.05	0.02	0.01	0.002	0.001	
One Tail		0.10	0.05	0.025	0.01	0.005	0.001	0.0005	
Tail Probabilities									

Appendix for Chapter 10

This Appendix provides instructions on setting up and expanding the use of the Experimenter in *FlexSim*.

Section 1 Setting up the Experimenter

When statistical distributions are used, simulation models are often run a number of times; these models are referred to as replications. These multiple simulation runs may be accomplished manually; however, the Experimenter feature of FlexSim can ease that burden. The Experimenter is located in the Statistics dropdown menu at the top of the screen. The default Experimenter interface is shown in Figure A.10.1. The tabs in the experimenter provide a procedure to establish the experimental runs.

The Experimenter provides a way to automatically run a series of simulation scenarios and collect data for comparison.

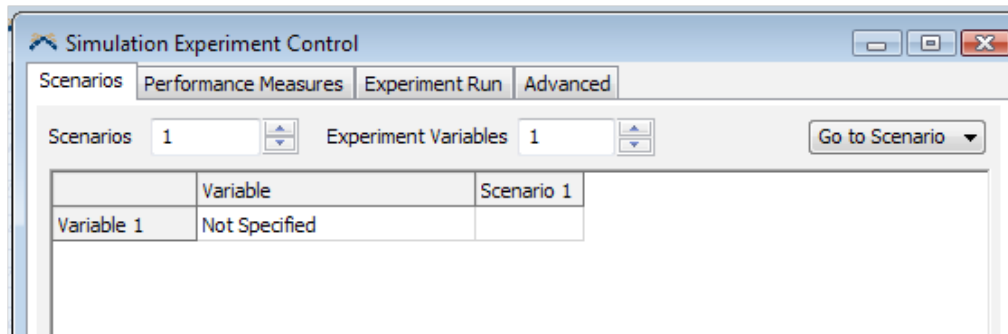


Figure A.10.1 Experimenter tab window

- *Scenarios*: Setup the number of scenarios and variables to be used
- *Performance Measures*: Define the variables that should be tracked during the experiments
- *Experiment Run*: Control the execution of the experiments and view results
- *Advanced*: Add custom logic to be executed during the experimental runs

Section 2 Defining scenarios and performance measures

If there is more than one scenario, and there is a desire to change the value of variables for each scenario, then those choices are made on the Scenarios tab window.

Clicking in the Not Specified cell opens a series of dialogues to specify the variable(s) to change. The first choice is to specify the type of variable to be changed. The choices include object label, object variable, table value, etc. The next choices depend on the first selection and will finalize defining the variable

For example, Figure A.10.2 illustrates the case where the capacity of a queue changes from scenario to scenario. To select this variable in the Experimenter, first select object variable, then choose the specific object, designate Queue 2, and finally select the specific variable/attribute, in this case, maxcontent from the provided list.

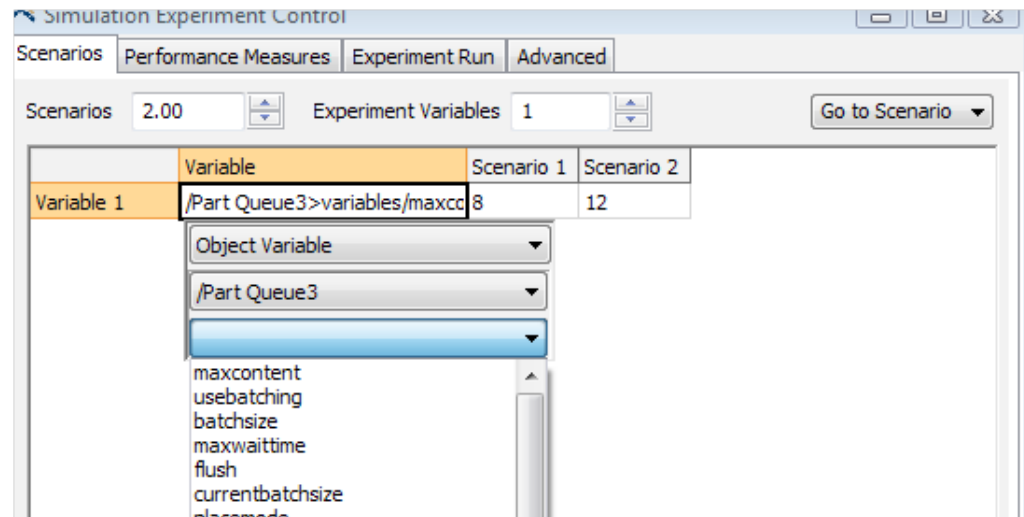


Figure A.10.2 Performance Measure tab

Performance Measures are defined variables that are important indicators of the simulation results. As shown in Figure A.10.3, the Performance Measure tab allows a number of statistics to be defined for each simulation run.

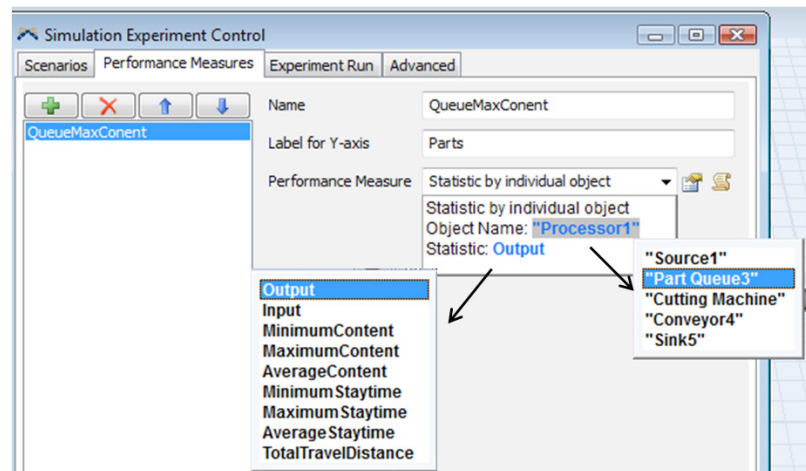


Figure A.10.3 Selecting an object variable to change

- Plus Sign: Creates a new Performance Measure.
- Name: Text field for naming the Performance Measure.

- Performance Measure: Drop-down list of possible types of statistics
 - Drop-down menu choice for Object Name – double clicking on the blue text brings up a list of choices that are selected by double clicking on the choice.
 - Drop-down menu choice for Statistic – same selection windows as for the object name

Every Performance Measure can be customized, or new measures created, through the code icon next to the performance measure.

Section 3 – Advanced experimenter options

The experiment execution is controlled on the Experiment Run tab as shown in Figure A.10.4. The length of run for each experiment is set as well as any warm-up time that should expire before statistics are calculated. These values should be in the same time units as the entire simulation. The number of replications for each scenario is also specified on this tab page.

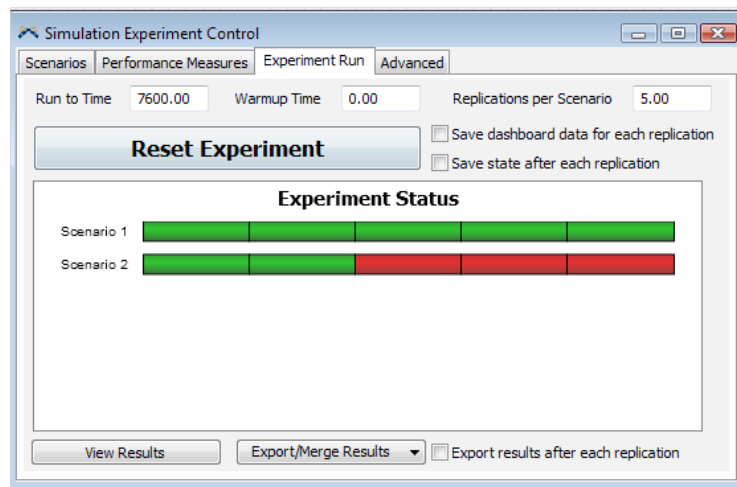


Figure A.10.4 Experimentation with five replications of two scenarios

The Run Experiment button begins the simulation execution. A progress bar shows the current status of the run. Since the experimental simulations are run without the animation the execution time is very fast. When the run is complete there are various options for analyzing the results. At the bottom of the screen, the View Results button provides graphical representations. Results can also be automatically expired after the entire experiment or after each replication. The state of the model after each replication can also be saved by checking the box “save state after each replication” in the top right corner of the page. The state are saved in the Flexsim Project folder and can be opened by choosing file/State Files/Load State on the Flexsim main tool bar.

The Advanced tab on the experimenter, as shown in Figure A.10.5, allows for even more specialized experimenter run control. It contains triggers for various events that occur during the execution of the experimenter. Default values or custom scripts can be added to each trigger.

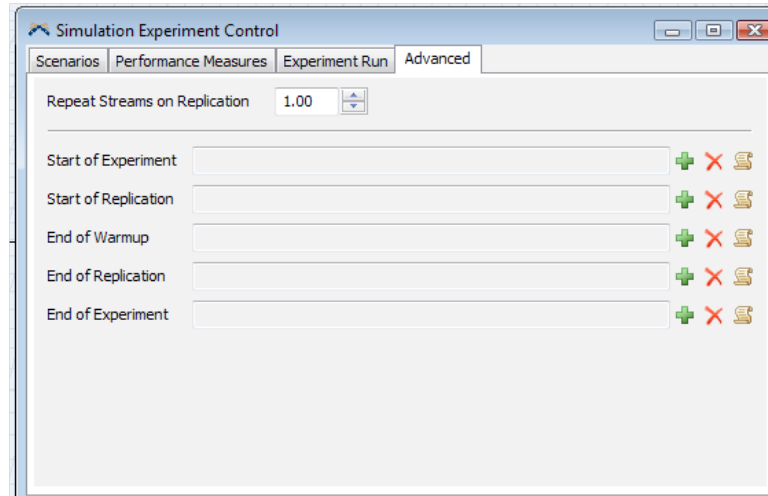


Figure A.10.5 Experimenter Advanced tab

Appendix for Chapter 11

This Appendix provides additional information on Exercise 11-1, Kegglers Brew and Exercise 11-2, Chairs for Tots.

Section 1 – Exercise 11-1 Kegglers Brew

Application Notes: (this is only one of many ways to build the simulation) An example model layout is provided in Figure A.11.1

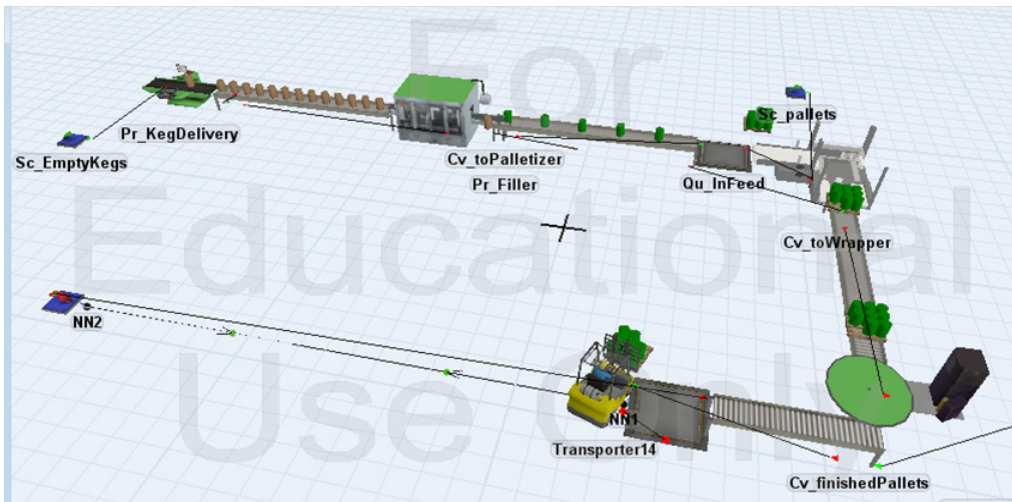


Figure A.11.1 Kegglers Brew with in-feed queue and new 3D objects

- Set the unit of time to one second.
- Use itemtypes for the following products: 1-Full kegs; 2-Quarter kegs; 3-Fridge Kegs.
- Use a source for kegs and pallets. Set their itemtypes to the product being run. Set the interarrival time to 0. This will ensure that the material is always available.
- Use a cylinder for the keg source, and set the item size accordingly in the OnExit trigger of the keg source.
- Create a global table of three rows and three columns. Rows represent product type, and columns represent the rates of the placement, filler, and pallet objects.

- Use the “update combiner component list” command on the input trigger of the combiner. Create a global table with one row and three columns. Each column is the number of kegs per pallet for a product.
- Build the model first without any breakdowns. Use the Stop time setting to determine the output for a period of time to see if it is correct.
- If the system is not capable of producing the smaller kegs, try adding an in-feed queue before the combiner. Delete the output from the conveyor to the combiner and route it through the queue. Set the queue to a max of 12.
- Change the 3D image of the filler, palletizer, and wrapper using images in the fs3d folder.
- Set the color of the kegs at the source or placement machine to some color. Change the color at the filler.
- Use network nodes between the pallet queue and sink at the end of the simulation. Set the virtual distance on the nodes and the speed on the truck to get the desired travel time.
- The Experimenter can be used by changing the itemtype of each of the sources for a scenario. This might be useful in order to replicate runs once the simulation is validated.

Results from a typical run

Product	Without in-feed queue			With in-feed queue	
	Max Pallets	Without Breakdowns	With Breakdowns	Without-Breakdowns	With Breakdowns
Full (1)	800	800	768	800	762
Quarter (2)	960	942	832	960	820
Fridge (3)	960	743	662	960	759

Figure A.11.2 Throughput - pallets per 8-hr. shift

Section 2 Exercise 11-2 Chairs for Tots

Application Notes: (this is only one of many ways to build the simulation) An example model layout is provided in Figure A.11.3.

- Each order can be considered a flowitem since all times reflect the order size.
- Use one minute for the time unit.
- Assume the simulation starts at 8 a.m. on Monday.
- Set up a Time Table to stop the orders at noon on Friday. Stop the simulation at 4 p.m. Friday (6240 minutes)

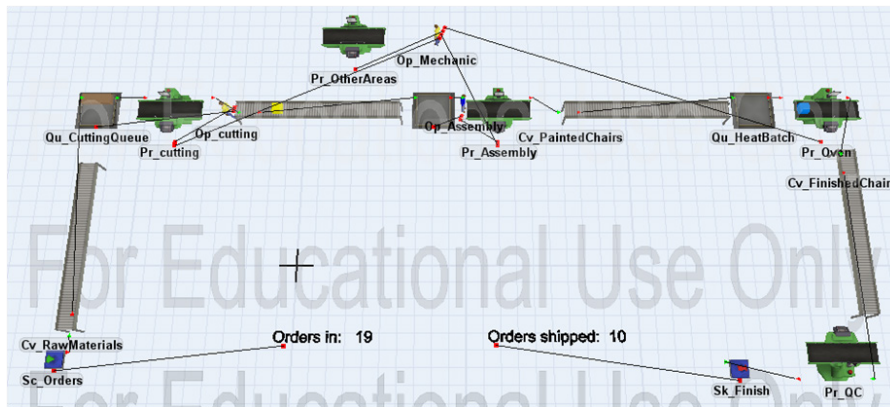


Figure A.11.3 Chairs for tots model

- The problem with the combiner is typical of actual production runs. At the end of the day there might not be enough pieces for the combiner to operate. Consequently, the pieces simply have to wait and are never processed.
 - A queue can be used as a batching object. Set the maximum content size and batch size to the same value.
 - The queue also provides for a maximum wait time. If too small of a wait time is used then the efficiency of batching is lost; too long and throughput can be reduced

In order to help validate the model, operate the simulation without any breakdowns and with the processing and order arrival times set as a constant at their average value. Run for a period of time and check the results.

- A processor without any inputs or outputs can be used to represent the plant.
- By default a breakdown hash no specific state identified so that the total simulation time will be used.
 - Carefully look at the down function for each breakdown entry.
 - Set the person to be called correctly. (see Figure A.11.4)
 - Set the pre-empt choice correctly :
 - ☐ Short-term machine breakdowns should pre-empt other operator tasks.
 - ☐ Long-term breakdowns do not pre-empt.
- Use the Experimenter to run at least 10 replications and, at a minimum, use work in progress at the end of each run as a performance measure, along with the total number of shipments.
- Try the following modifications to the operations:
 - Add another maintenance person.

- Do not use transport between the cutter queue and the cutter.
 - Experiment with various wait times going into the oven as well as different batch sizes.
 - Use a continuous five-lane oven.

Typical results for 10 replications of a one-week operation:

	Avg. orders shipped	Avg. orders in-process
Current operations	195	9.1
Wait time of 90 min	197	7.6
Wait time of 90 min and automated cutter in-feed	199	0.6

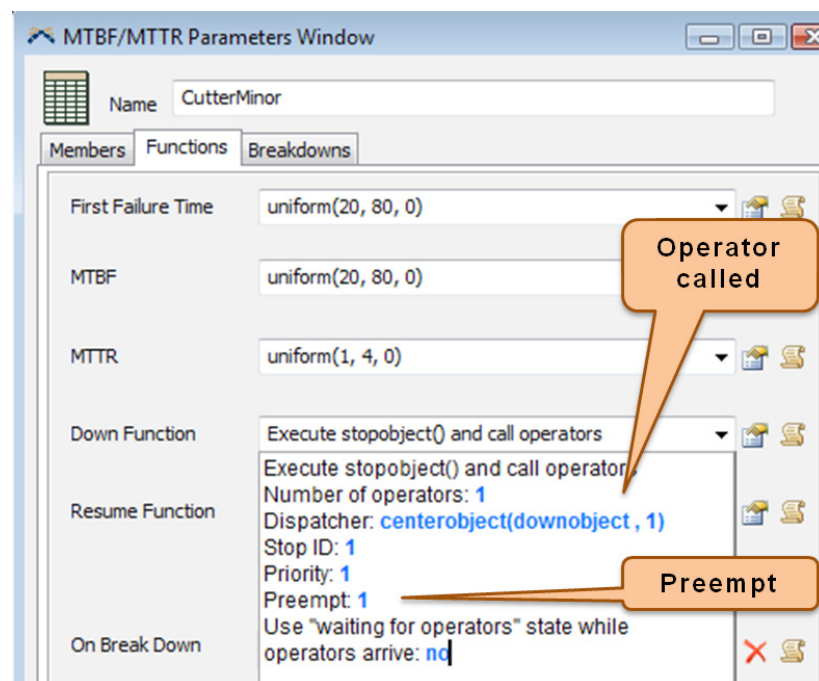


Figure A.11.4 Setting up an MBTF table to use an operator for service

Appendix for Chapter 12

This Appendix provides a summary of commonly used *Flexscript* commands, a discussion of the *FlexSim* software architecture, and additional information on Exercise 12-1, Hilltop Steel Production.

Section 1 Commonly-used commands

The following commands and access variables are used in referencing objects in *FlexSim*:

The current and item references:

- `current` - the current resource object. It is often an access variable in picklists.
- `item` - the involved item for a trigger or function. It is often an access variable in picklists.

Referencing

Command(parameter list)	Explanation	Example
<code>first(node)</code>	This returns a reference to the first ranked object inside of the node.	<code>first(current)</code>
<code>last(node)</code>	This returns a reference to last ranked object inside of the node.	<code>last(current)</code>
<code>rank(node,ranknum)</code>	This returns a reference to the object at a given rank inside the node.	<code>rank(current,3)</code>
<code>inobject(object,portnum)</code>	This returns a reference to the object connected to the input port number of the node.	<code>inobject(current,1)</code>
<code>outobject(object,portnum)</code>	This returns a reference to the object connected to the output port number of the node.	<code>outobject(current,1)</code>
<code>centerobject(object,portnum)</code>	This returns a reference to the object connected to the center port number of the node.	<code>centerobject(current,1)</code>
<code>next(node)</code>	This returns a reference to the next ranked object of the node.	<code>next(item)</code>

Object Attributes

Command(parameter list)	Explanation
getname(object)	This returns the name of the object.
setname(object,name)	This sets the name of the object.
getitemtype(object)	This returns the itemtype value of the object.
setitemtype(object,num)	This sets the itemtype value of the object.
setcolor(object,red, green, blue)	This sets the color of the object.
colorred(object) green, blue, white...	This sets the color of the object to red, blue, green, white, etc.
setobjectshapeindex(object,indexnum)	This sets the 3D shape of the object.
setobjecttextureindex(object,indexnum)	This sets the 3D texture of the object.
setobjectimageindex (object,indexnum)	This sets the 2D texture of the object. This usually only applies if you are using the planar view.

Object Spatial Attributes

Command(parameter list)	Explanation
xloc(object) yloc(object) zloc(object)	These commands return the x, y, and z locations of the object.
setloc(object,xnum,ynum,znum)	This sets the x, y, and z location of the object.
xsize(object) ysize(object) zsize(object)	These commands return the x,y, and z size of the object.
setsize(object,xnum,ynum,znum)	This sets the x, y, and z size of the object.
xrot(object) yrot(object) zrot(object)	These commands return the x,y, and z rotation of the object.
setrot(object,xdeg,ydeg,zdeg)	This sets the x, y, and z rotation of the object.

Object Statistics

command(parameter list)	Explanation
content(object)	This returns the current content of the object.
getinput(object)	This returns the input statistic of the object.
getoutput(object)	This returns the output statistic of the object.
setstate(object,statenum)	This sets the current state of the object.
getstatenum(object)	This returns the current state value of the object.
getstatestr(object)	This returns the current state of the object as a string.
getrank(object)	This returns the rank of the object.
setrank(object,ranknum)	This sets the rank of the object.
getentrytime(object)	This returns the time the object entered the object it is currently in.
getcreationtime(object)	This returns the time the object was created.

Object Labels

command(parameter list)	Explanation
getlabelnum(object,labelname) getlabelnum(object,labelrank)	This returns the value of the object's label.
setlabelnum(object,labelname ,value) setlabelnum(object,labelrank ,value)	This sets the value of the object's label.
getlabelstr(object,labelname) getlabelstr(object,labelrank)	This gets the string value of the object's label.
setlabelstr(object,labelname ,value) setlabelstr(object,labelrank ,value)	This sets the string value of the object's label.
label(object,labelname) label(object,labelrank)	This returns a reference to the label as a node. This command is often used if you have a label that is used as a table.

Tables

command(parameter list)	Explanation
gettablenum(tablename,rownum,colnum) gettablenum(tablenode,rownum,colnum) gettablenum(tablerank,rownum,colnum)	This returns the value in the specified row and column of the table.
settablenum(tablename,rownum,colnum,value) settablenum(tablenode,rownum,colnum,value) settablenum(tablerank,rownum,colnum,value)	This sets the value in the specified row and column of the table.
gettablestr(tablename,rownum,colnum) gettablestr(tablenode,rownum,colnum) gettablestr(tablerank,rownum,colnum)	This returns the string value in the specified row and column of the table.
settablestr(tablename,rownum,colnum,value) settablestr(tablenode,rownum,colnum,value) settablestr(tablerank,rownum,colnum,value)	This sets the string value in the specified row and column of the table.

settablesize(tablename,rows,columns) settablesize(tablenode,rows,columns) settablesize(tablerank,rows,columns)	This sets the size of the table in rows and columns.
gettablerows(tablename) gettablerows(tablenode) gettablerows(tablerank)	This returns the number of rows in the table.
gettablecols(tablename) gettablecols(tablenode) gettablecols(tablerank)	This returns the number of columns in the table.
clearglobaltable(tablename) clearglobaltable(tablenode) clearglobaltable(tablerank)	Sets all number values in the table to 0. Also clears string data.

Object Control

command(parameter list)	Explanation
closeinput(object)	This closes the input of the object.
openinput(object)	This re-opens the input of the object.
closeoutput(object)	This closes the output of the object.
openoutput(object)	This re-opens the output of the object.
sendmessage(toobject,fromobject, parameter1,parameter2,parameter3)	This causes the OnMmessage trigger of the object to fire.
senddelayedmessage(toobject,delaytime,fromobject, parameter1,parameter2,parameter3)	This causes the OnMessage trigger of the object to fire after a certain delay time.
stopobject(object,downstate)	This tells the object to stop whatever its operation is and go into the given state.
resumeobject(object)	This allows the object to resume whatever its operation is.
stopoutput(object)	This closes the output of the object, and accumulates stopoutput requests.
resumeoutput(object)	This opens the output of the object once all stopoutput requests have been resumed.
stopinput(object)	This closes the input of the object, and accumulates stopinput requests.
resumeinput(object)	This opens the input of the object once all stopinput requests have been resumed.
insertcopy(originalobject,containerobject)	This inserts a new copy of the object into the container.
moveobject(object,containerobject)	This moves the object out of its current container into its new container.

Object Variables

command(parameter list)	Explanation
getvarnum(object,variablename)	This returns the number value of the variable with the given name.
setvarnum(object,variablename,value)	This sets the number value of the variable with the given name.
getvarstr(object,variablename)	This returns the string value of the variable with the given name.
setvarstr(object,variablename,string)	This sets the string value of the variable with the given name.
getvarnode(object,"variablename")	This returns a reference to the variable with the given name as a node.

Prompts and Printouts

command(parameter list)	Explanation
pt(text string)	Prints text to the output console.
pf(float value)	Prints a floating point value to the output console.
pd(discrete value)	Prints an integer value to the output console.
pr()	Creates a new line in the output console.
msg("title","caption")	Opens a simple Yes, No, Cancel dialog.
userinput(targetnode,"prompt")	Opens a dialog box where you can set the value of a node in the model.
concat(string1,string2,etc.)	This returns the string concatenation of two or more strings.

Section 2 *FlexSim* software architecture

The functionality of a simulation application is directly tied to its software language and underlying architecture. Capabilities such as visualization, ease of use, and compatibility with other applications require a modern structure. The *FlexSim* software structure allows it to easily facilitate virtual reality modeling, design of experiments, result analysis, data collection, system optimization, and user creation of applications complete with powerful user interfaces. *FlexSim* supports C++, *OpenGL*, and provides its own language scripting capability. *FlexSim* is architected as an extensible, modular, and open (or closed) source platform.

FlexSim is developed using object-oriented methods and C++, with a proprietary database abstraction layer. It is also interoperable with external *SQL* databases. *FlexSim* runs on the *Microsoft Windows* operating system and has a bi-directional interface to *Microsoft's Visual Studio*. *FlexSim's* distributed simulation application, *FlexSim DS*, has an extra network layer that facilitates managing data transfer between

FlexSim stations running on different computers and/or on the same computer. This architecture provides a secure, robust, and extensible system for managing compute-intensive simulation studies both locally or across a network. A high-level view of the main application architecture is shown Figure A.12.1.

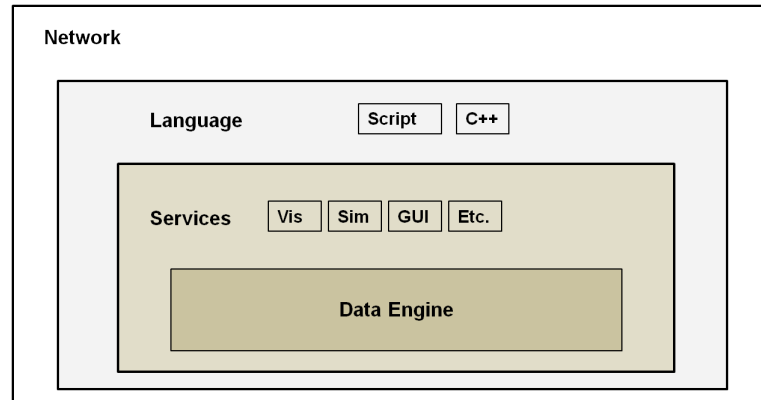


Figure A.12.1 High-level view of *FlexSim*'s architecture

FlexSim's main components include the following:

- Data Engine
- Simulation Engine
- User Interface Engine
- Language Engine/Services
- Visualization/VR Engine
- MS Visual Studio Interface
- Network/Internet Layer (FlexSim DS)

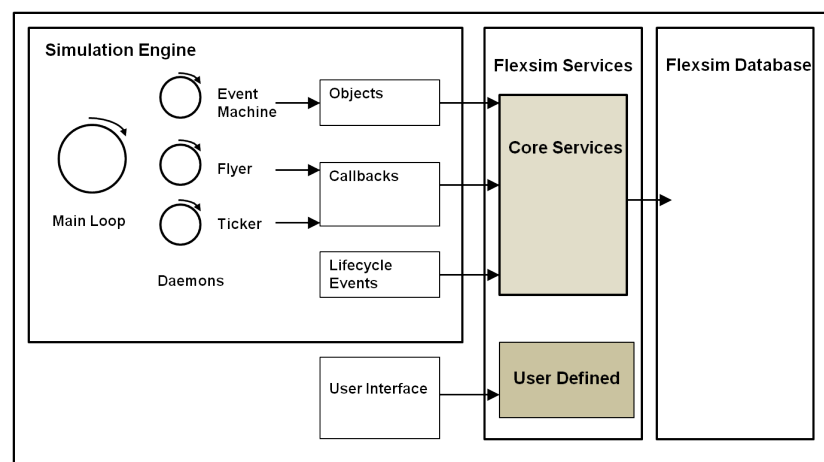


Figure A.12.2 Interoperation of Main Components

Each of these components is briefly explained in the following subsections. The interoperation of these components is illustrated in Figure A.12.2. When a model runs and/or there is interaction with the simulation application, simulation events and daemon events, user interface events and network events, trigger functions which utilize the Services library module. Those functions in turn modify the *FlexSim* database directly.

Data engine

The data engine, as illustrated in Figure A.12.3, provides a persistent, robust, and high-speed data management system that maintains a central repository of all of the data that is used in a simulation, including model states, experimentation, user interfaces, application data, etc. The principal motivations for the use of a dedicated proprietary data management system are centralization of data, persistence of data, and fast access. The data engine provides a unified binary data representation for RAM, disc, and network transmission. There is also the ability to exchange data in XML format.

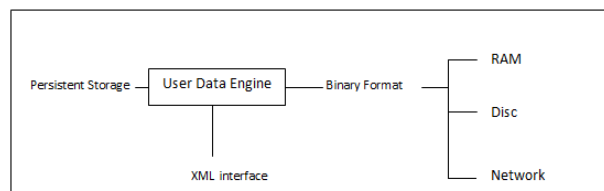


Figure A.12.3 Data Engine

Simulation engine

The simulation engine, as illustrated in Figure A.12.4, is a core component for running and managing simulation models and experiments. Its primary functions include the following:

- Daemon-driven discrete-event simulation and virtual reality events
- Customization and extension for user-designed events in real and simulated time
- Synchronization of discrete-event model events and real-time events
- Simulation experimentation functions

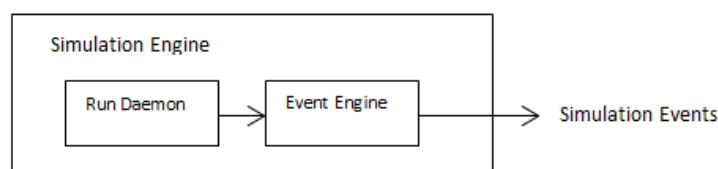


Figure A.12.4 Simulation Engine

User interface engine

The user interface engine, as shown in Figure A.12.5, enables the design, editing, storage and deployment of user interfaces. User interfaces are stored in the *FlexSim* database and are thus part of a simulation application.

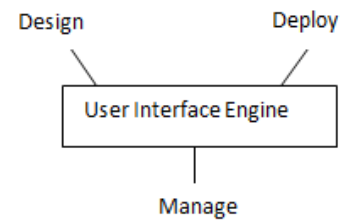


Figure A.12.5 Interface Engine

Language engine/services

This component provides an extensible base of functionality for conducting simulation studies and for general programming. As shown in Figure A.12.6, the language engine enables dual language execution of both native C++ and *FlexSim*'s scripting language, *Flexscript*. It incorporates the base library of functions for creating and running simulation models and experiments, as well as creating simulation applications/environments. It includes a built-in script parser, a dual-language executor, an interface to Microsoft *Visual Studio*'s IDE, and a binary-level interface to C++.

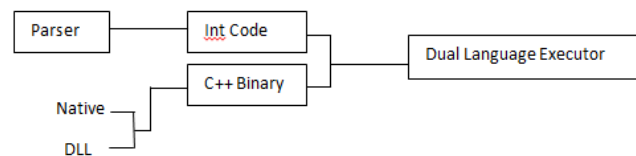


Figure A.12.6 Language Engine

Visualization/VR engine

The OpenGL-based visualization engine provides 3D visualization and virtual reality (VR) functionality for high-fidelity simulation and visual effects. It contains C++ and scripting programming interfaces, interfaces with common 3D model formats, enables real time audio play and manipulation, provides support for DirectInput and other 3D input devices, and provides functionality for user feedback about interactions with the visual scene to facilitate extensibility of user interaction.

Network/Internet layer

As shown in Figure A.12.7, the network interface layer allows communication and data exchange between instances of *FlexSim* running across a Network via TCP/IP. This component facilitates operation across the Internet. *FlexSim*'s open architecture allows any of the core modules of *FlexSim* to have access to the network layer. This access facilitates interaction between *FlexSim* stations and allows many possible configurations of simulation models and experiments, user interfaces, VR environments, etc., across a network.

The network layer allows advanced applications to make use of

- parallel processing (the execution of multiple linked simulation models simultaneously);

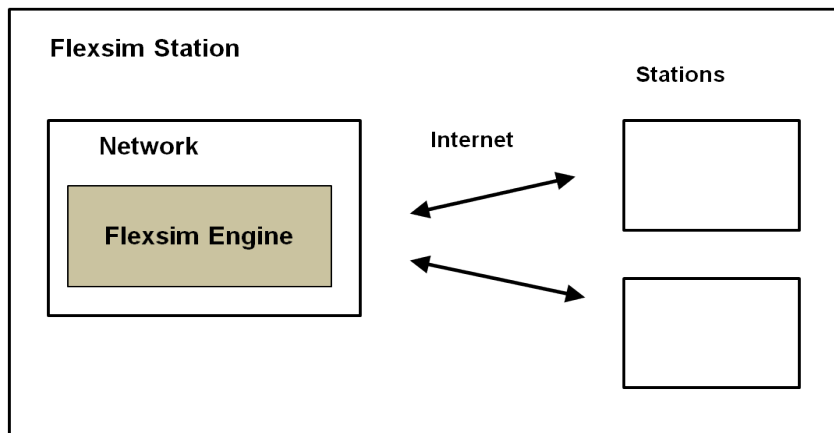


Figure A.12.7 Network Configuration

- multi-user collaboration and interaction across the Internet via satellite FlexSim stations logged into a central FlexSim server station;
- distributed models (the linking of multiple resource-intensive *FlexSim* submodels to form a coordinated main model and thus allow much larger models);
- remote model viewing and interaction across the Internet by logging in from a remote FlexSim client station into a FlexSim server station.

Section 3– Exercise 12-1 Hilltop Steel

Application Notes: (this is only one of many ways to build the simulation) An example model layout is shown in Figure A.12.8.

- Choose one second as the unit of time.
- Create a global table with one row and three columns. The three columns contain the operating rate (bolts/hr), the regular speed cycle time (sec), and the high-speed cycle time (sec), respectively.
- Use labels on the source and on processors that contain the current speed values. In the rate triggers, set the code to get the label value.
- Use the OnReset triggers to initialize the label values.
 - For the source; OnReset trigger, get the correct global table value for the production rate, calculate the appropriate cycle time and store it in the source label. Be careful to check that the calculation doesn't divide by 0.
 - For the processor; OnReset triggers, simply move the normal cycle time from the global table to the label.
- Use visual tools to display the machine speeds, production rate, and source blockage percent.

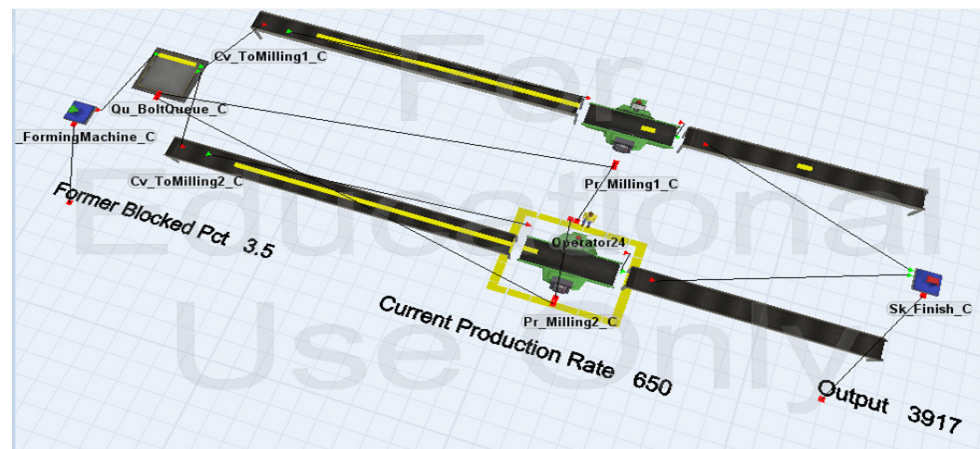


Figure A.12.8 Hilltop Steel simulation

- Set various production rates and check that the correct amount of material is produced.
- Add breakdowns. Set up the experimenter to run five replications for the one scenario, for a total of 40 hours (144,000 sec).
- Prepare the logic for dynamically changing the machine speeds.
- Use the queue flow trigger to add logic. Open the code icon and clear the template code. Add logic after the base variable definitions.
 - Variable values needed:
 - ☐ Current state of milling machines 1 and 2.
 - ☐ Flags indicating if high speed is turned on (add another label to the processors): 0 = normal speed; 1 = high speed. Initialize the flag value to 0 in the reset trigger
 - Create center connections between the queue and each processor
 - Logic conditions:
 - ☐ If a processor is down (state 11) and the second processor is at normal speed (flag = 0) then change the cycle time label on the second processor to the fast speed and set the flag to 1.
 - ☐ In the same way, return a processor to normal speed by checking to see if a processor is not down and the second processor is at high speed.

The code could look like the following, added to the flow trigger on the queue:

```
/**Custom Code*/

treenode item = parnode(1);

treenode current = ownerobject(c);
```

```

treenode tempobject;

int curmincontent = 1000000000; // this sets the integer to the
largest possible value that an integer can hold.

double curminindex = 0;

// start speed control code

treenode finish1 = centerobject(current,1);
treenode finish2 = centerobject(current,2);

int speed1 = getlabelnum(finish1,"HighSpeed");
int speed2 = getlabelnum(finish2,"HighSpeed");
int state1 = getstatenum(finish1);
int state2 = getstatenum(finish2);

double highspeed = gettablenum("Operations",1,3);
double stdspeed = gettablenum("Operations",1,2);

if(state1 == 11 && speed2 == 0)
{
    setlabelnum(finish2,"CycleTime", highspeed);
    setlabelnum(finish2,"HighSpeed", 1);
}

if(state1 != 11 && speed2 == 1)
{
    setlabelnum(finish2,"CycleTime", stdspeed);
    setlabelnum(finish2,"HighSpeed", 0);
}

if(state2 == 11 && speed1 == 0)
{
    setlabelnum(finish1,"CycleTime", highspeed);
    setlabelnum(finish1,"HighSpeed", 1);
}

if(state2 != 11 && speed1 == 1)
{
    setlabelnum(finish1,"CycleTime", stdspeed);
    setlabelnum(finish1,"HighSpeed", 0);
}

// end speed control code

```

- Use the Experimenter to run replications of 40 hours.
- Use visual tools to display the machine speeds and production rate.

Typical output average values obtained after 40 hours of simulation are provided below. Efficiency represents the number of bolts produced during the period divided by the number of bolts that should have been produced based on the production rate.

Without dynamic speed control:

Target Rate/hr	Efficiency %	Former Blocked %
600	97	2.4
650	92.5	7.4
700	86.3	13.5

With dynamic speed control:

Target Rate/hr	Efficiency %	Former Blocked %
600	99.5	0.5
650	98	1.8
700	93	6.8

Appendix for Chapter 13

This Appendix provides details on the Recorder object in *FlexSim* and additional information on Exercise 13-1, Fister's Express, and 13-2 Peoples Surgery Center.

Section 1 Recorder details

The recorder option “User-defined data” from the Type of Data picklist allows the graphing of specific information collected during the run of a simulation. There are several choices to make concerning the type of output to display (line graph, histogram, etc). The type of display should be consistent with the type of data being collected. The data that will be recorded is identified by using the browse button (...) and choosing a node from the tree view.

For example, the Figure A.13.1 shows how the recorder was set up to graph the speed of the machine shown in the chapter example during its startup cycle.

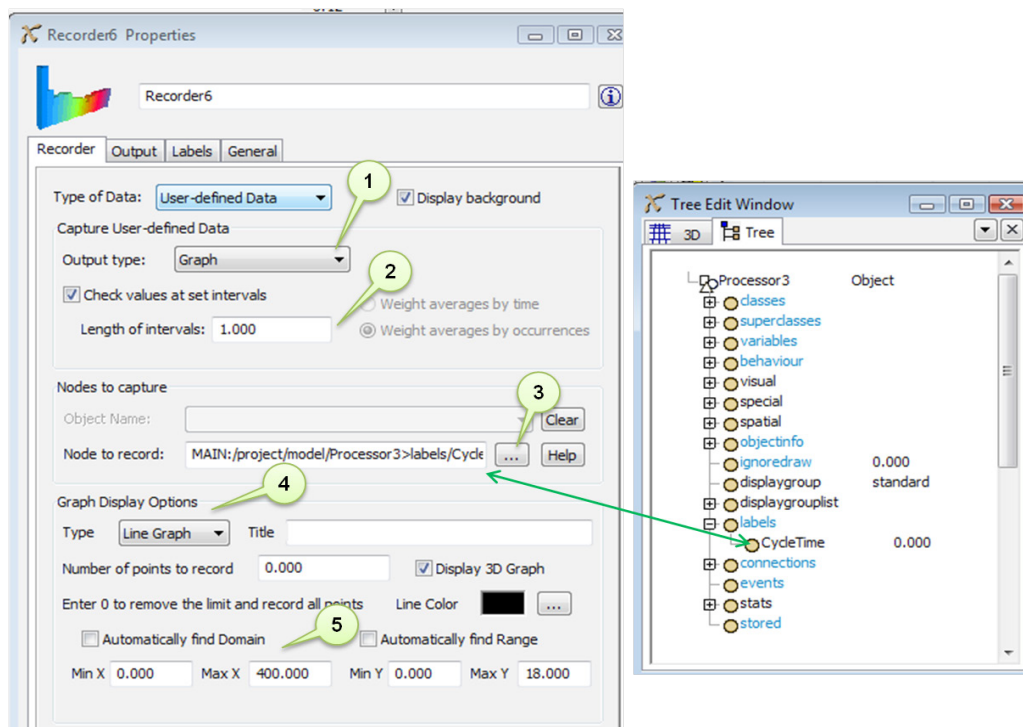


Figure A.13.1 Setting up a line graph using the recorder

The steps to follow to set up the recorder following the selection of “User-defined data” as the chart type are as follows:

1. Set the output type to Graph.
2. Select the box to check values at set intervals; this affects the resolution. Set the interval to 1; not checking the box will have the recorder updating every calculation cycle.
3. To select a node to capture, use the Browse icon (...), which opens a tree view.
 - Dive into the object’s data subnodes to find the one to record (in this case the labels/cycletime node).
 - Click on the Select button to inset the path into the “Node to record” field.
4. Select the Graph Display Options.
 - Select a line graph.
 - Put a zero in the Number of points to record field; this means an unlimited number.
5. The x and y axis can be automatically established as the simulation progresses or they can be set manually.

Section 2 – Exercise 13-1 Fister’s Express

Application Notes: (this is only one of many ways to build the simulation.)

- Use one minute as the time unit.
- Treat an order as a combination of five entries. No need to simulate the separate items in each of the categories.
- Create two new items in the flowitem bin:
 - An order pallet with a label-table and an order ID label; note that individual labels could also be used for each category, but it makes referencing them later more difficult.
 - A category box with labels for the number of items and the category/part type.
- On the source:
 - Use the order pallet
 - On the OnExit trigger, fill out the label-table on the pallet with the appropriate order quantities.
- Use a queue to hold the orders as they are created.

- On the queue entry trigger set the itemtype sequentially by setting it to the current input total of the queue—for example, `getinput(current);`
- Set up another queue to control the production.
 - Set the Maximum Content value to 1 and close the output on reset.
 - Add a label as a counter flag to check when picking is finished.
 - When a pallet enters the production queue
 - Send a message to each picking robot—send the number of items to be picked and the order ID as variables. Also sending the category number (e.g., 1 for steak) is optional but could eliminate the need to make sure each picking processor knows its number to look up the picking times.
 - Set the counter flag label to 5 (number of robots to reply).
 - Each time a message is received from a robot
 - Decrease the counter flag.
 - Check to see if the picking is complete; if it is complete, open the queue output.
 - Reset the counter flag to the new total.
 - When a pallet exits, close the queue output
- All of the picking robots can be set up as processors.
 - Create labels for the number of items, order ID, and cycle time.
 - On reset close the input.
 - When a message is received
 - Use the search time table to calculate the pick time; note that it may be 0 if no items are requested.
 - Set label values on the processor for the cycle time, order ID and number of items.
 - Open the input ports.
 - On exit
 - Send a message to the control queue.
 - Close the input ports.

- On the flowitem, set the labels with the number items and the order ID.
- On the combiner
 - Make sure the pallet is on input port 1.
 - Set the combiner to get one item from each input 2-6.
 - Set the processing time.
 - On input , for the variable “port” greater than 1:
 - Check for the correct number of items.
 - Get the order quantity from the label table on the pallet which is always the first item in the combiner.
`gettablenum(label(first(current),”labelname”),portnumber,1);`
 - Check against the number on the box label
- On the sink
 - Set labels for the current order time in system, the total order time in system, and the average order time in system.
 - On entry
 - Calculate the time in process for each order.
 - On entry, get the creation time of the pallet with the command `getcreationtime(item)`
 - Note that the pallet is considered the item, and the boxes are only sub items to the pallet.
 - Set the labels accordingly.
- Visualizations
 - On exit from the robot pickers, the boxes can either be given a color showing which category they belong to or given a color based on their order ID using the command `colorarry(item, ordered);`
 - A box with an order quantity of 0 could be colored white or made to “vanish” by using the command `switch_hideshape(item,1);` The box will not be visible but will still be treated as a flow item in the simulation.
 - Use a recorder set up as a histogram collecting data from the label on the sink that has the current time in system for the item.

The simulation may look like the layout shown in Figure A.13.2.

Sample results:

Order Cycle time (min)	9	10
Orders Shipped	528	934
Avg. time in system (min)	248	67
Back Log	45	8

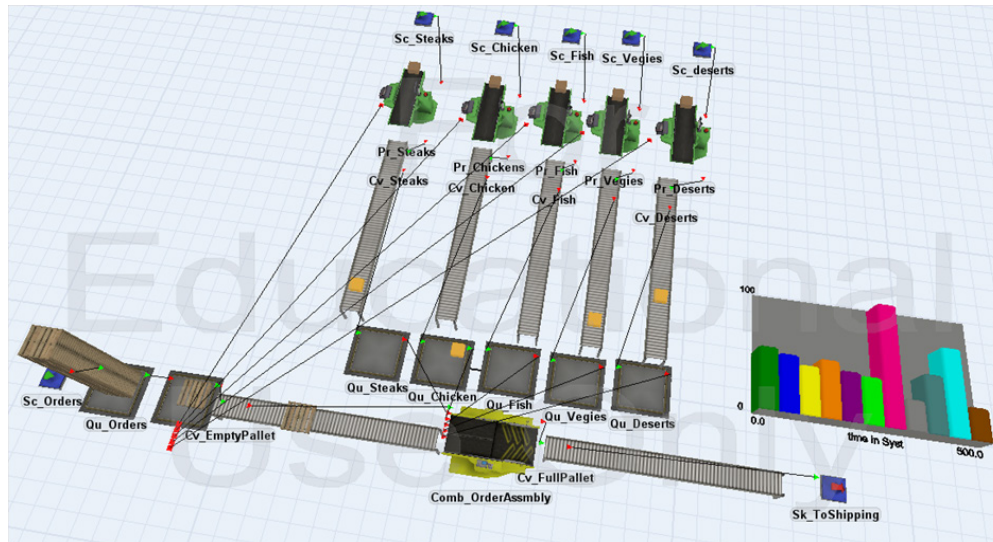


Figure A.13.2 Fister's Express

Section 3 – Exercise 13-2 Peoples Surgery Center

Application Notes: (this is only one of many ways to build the simulation.) A possible model layout is shown in Figure A.13.3.

```

/**Calculate speed*/
treenode item = parnode(1);
treenode current = ownerobject(c);

// calculate distance to lab
double xx = xloc(outobject(current,1));
double yy = yloc(outobject(current,1));
double xstart = xloc(current);
double ystart = yloc(current);
double xdists = xstart - xx;
double ydist = ystart - yy;

double dist = sqrt(xdist*xdists + ydist*ydist);
double speed = dist/maxof(0.0001, uniform(2,5,2));
return speed

```

Figure A.13.3 Possible script to calculate distance between flow nodes

- Use itemtypes for patients.
- Use two sources for patients
 - Use one source for before 10 a.m. and one for after.
 - Use a time table to control when the sources are active.
 - Assume the simulation will start at 7 a.m. and end at 7 p.m.
 - Set itemtypes in each source using a dempirical table.
 - Create a label on the “person” flow item as a flag for lab tests being completed.
 - Create labels on one source to count the number of patients. Update it each time an itemtype is set. Connect sources by a center connection for easier specifying.
- Use the Maximum Content on the admission processor to indicate the staff levels.
 - Check for lab flag to see if a patient is coming back from the lab or not.
- Run and validate the simulation model to make sure the patients are entering correctly.
- Have two outputs for the surgery waiting area— one sink for surgery patients and one sink for patients who had lab work first.
- Set up the Orderlies (task executers) to take patients to surgery
 - Use network nodes with virtual distance between the surgery waiting queue and the surgery sink.
 - On the OnContinue trigger of the network node, calculate and set the maximum speed variable of the orderly (traveler). Do this for the network node at the surgery waiting areas and also for the surgery sink.
 - On reset, in the patient waiting area calculate the distance to surgery.
 - In the OnResourceAvailable trigger of the orderly, select the choice to return to home.
- For the lab
 - Use flow nodes between the admissions desk going to the lab and from the lab to both the waiting area and exit.
 - Calculate the speed for the patient by using the commands xloc, yloc to get the coordinates of the nodes. See Figure A.13.4.

- Have the check-in desk at the lab pull patients with a label flag that indicates they have not had lab work done. Set the flag appropriately when exiting from the registration desk.
- Have the lab pull patients with a flag value showing they have registered at the lab desk.
- On the lab waiting queue, select to use a transport but use the conditional transport choice drop-down
- Connect a flow node to the lab. From the flow trigger on the flow node, use logic to send patients back to either the waiting area or exit.

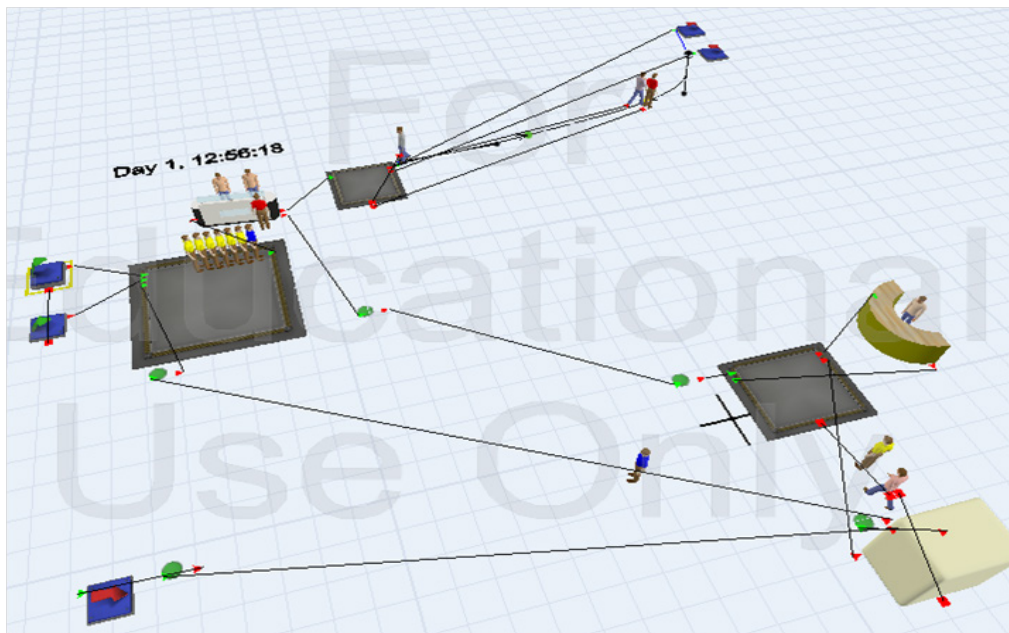


Figure A.13.4 Peoples Surgery Center model

- Calculate on each sink the time the patient was in the system using labels for total time and average time:
 - Use the stats_creation time node. Get its value using the command `getnodedenum(stats_creationtime(item))`.
 - Update both the labels for total time and average time.
- Once the simulation is validated, work on visualization:
 - Use the `setloc` command on the load trigger of the orderly to place the patient correctly. Do the same for the lab technician.
 - Use a cube or imported file for the main and lab registration desks. Use the `setloc` command to have patients stand in front of the desks. Uncheck the box to convey items across processor.
 - Add a time of day clock using a visual tool.

- Use the experimenter to run replications.
 - Run each replication for 12 hours.
 - Collect data on the number of patients and wait times.

Typical results for 20 replications using an interarrival time of 12.

Patient Type	Average Number	Avg Time in system (min)
Surgery Only	25.4	29.2
Lab and Surgery	3.6	115
Lab only	13.2	133

Average wait time – Main (min)	19
Average wait time – Surgery (min)	0.66
Average wait time – Lab (min)	11.90
Patients in area after 7 p.m.	0

Appendix for Chapter 14

This Appendix provides additional information on Exercise 14-1, James Peanuts, and Exercise 14-2, Western Grain.

Section 1 – Exercise 14-1 James Peanuts

Application Notes: (this is only one of many ways to build the simulation)

- Use one second as a unit of time—the size of the blender requires it.
- Use fluid to represent the peanut flow.
- Use an item-to-fluid object to change bags into fluid.
- Use an operator to carry bags to the item-to-fluid object.
- Make sure the inputs to the mixer are consistent with the Recipe table.
- Make sure the inputs to the blender are consistent with the Recipe table.
- Create a global table to define the main variables, such as dry peanut rate and use tank level marks. Set these variables using reset triggers.
- Use-tank logic
 - Start with the input to the use-tank closed.
 - Use the Low and High level mark triggers.
 - When falling through the MidMark level, open the tank input.
 - When rising through the HighMark level, close the tank input.
- Set a different color for each batch mixer component as well as the mixer itself.
- Performance metrics include the following:

- Use a Recorder object to watch the use tank level.
- Level displays can be modified using the FluidLevelDisplay tab on the tank object.
- Create a label on the mix tank to count the number of times it becomes empty.
- Improvement areas include the following:
 - Note that the increase in batch time caused by not being able to transfer all of the material at a time to the use tank was not considered.
 - Lowering the difference between the mid and high marks will help a little.
 - Create new logic on the use tank PassingHighMark trigger to shorten the time to empty the batch mixer.
 - When the level rises, close input.
 - If the amount of material left to transfer from the mixer is less than the space between the high and mid marks, change the mid mark to a new level. This will allow a transfer earlier than waiting for the original mid mark.
 - If the amount of material left to transfer from the mixer is greater than the space between the high and mid marks, make sure the mid mark is in its original position
 - There are many variations on this logic that can be used.

Possible control code on the use tank PassingHighMark trigger:

```

/**Midmark Control*/
treenode current = ownerobject(c);
int mode = parval(1); // 1 = rising, 2 = falling
#define rising 1
#define falling 2
#define either mode
if (mode == 1) // do this only if level rising through mark
{
    closeinput(current);
    // do the following logic if new control flag = 1
    if(gettablenum("Operations",2,1) == 1)
    {
        double mixercontent =
getvarnum(inobject(current,1),"curcontent"); // amount left in mixer
        double midmark = getvarnum(current,"midmark");
        double highmark = getvarnum(current,"highmark");
        double spaceavail = highmark - midmark; // normal control
space available
    }
}

```

```

if(mixercontent <= spaceavail)      // don't have to wait as long
    midmark = highmark - mixercontent;    //reset mid mark
else
    midmark = gettablenum("Operations",4,1);
    setvarnum(current,"midmark", midmark);
}

return 1;

```

Note that a variable flag was created on the global table to check if the new control logic should be used. Figure A.14.1 provides an example model for the exercise.

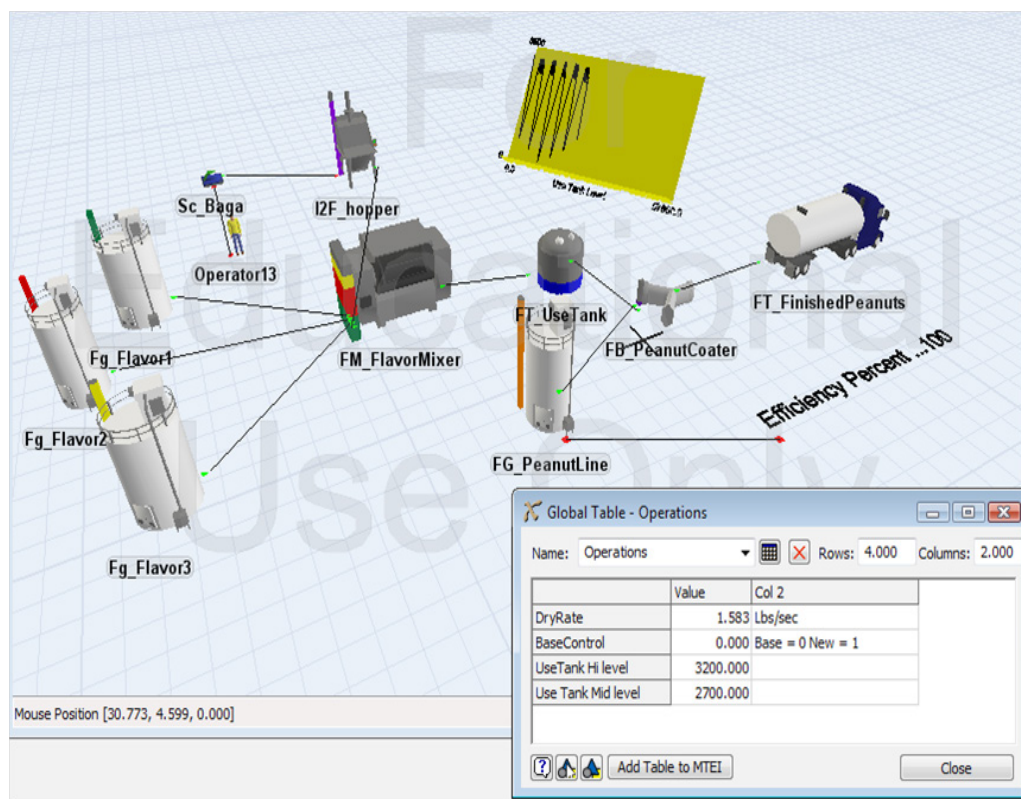


Figure A.14.1 James Peanut model

Section 2 – Exercise 14-2 Western Grain

Application Notes: (this is only one of many ways to build the simulation)

- Use the textured box flowitem for a barge.
- Use a transporter for the tug (open the student library).

- Set the location of the barge appropriately in the tug.
- Use a fluid tank for the loading hopper.
- Use a queue for a dock.
- Use message logic for loading/unloading operation delays and for setting loading values.
- Use network nodes for tugs to travel.

A possible model layout for the exercise is provided in Figure A.14.2.

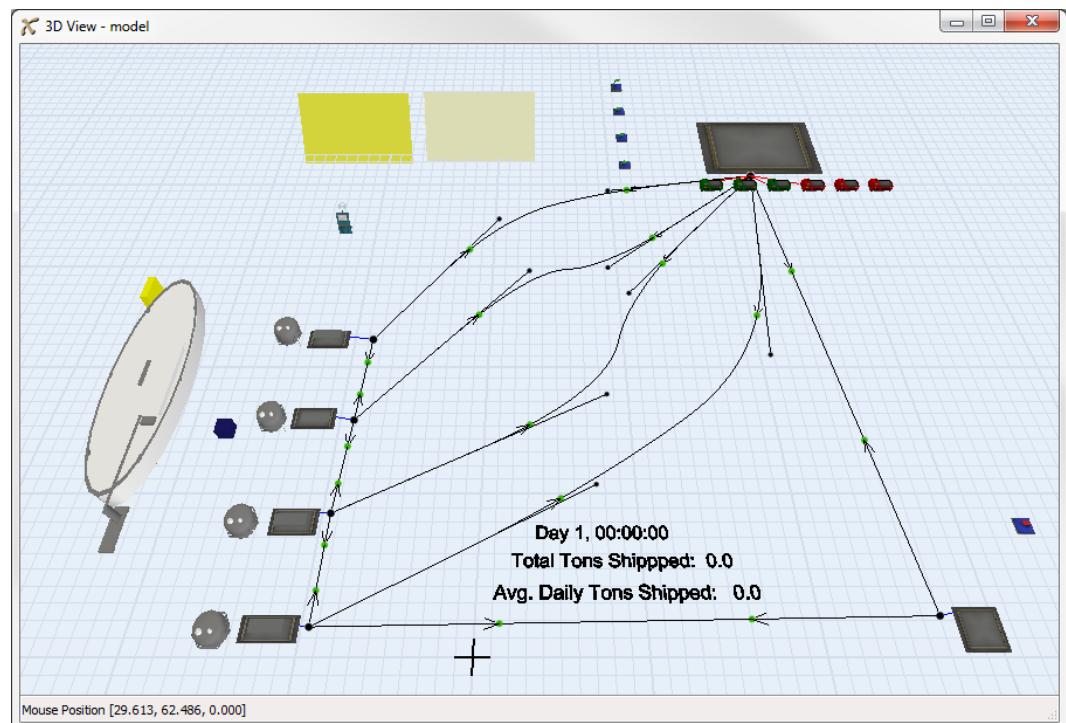


Figure A.14.2 Wheat Barge model

Appendix for Chapter 15

This Appendix provides additional information on Exercise 15-1, Custom Shapes.

Section 1 - Custom Shapes Exercise

Application Notes: (this is only one of many ways to build the simulation)

- Select 1 second as the base time unit.
- Set up data tables.
 - Operations—normal stamping speed and new configuration speed
 - Products—product names (row will be the ID number) with loss percent
 - Changeover—changeover times
 - Schedules—one for base schedule, one for new configuration
- Build one line and test it out.
 - Use one stamping processor.
 - Set a label on the stamping processor for the product ID number.
 - Get the cycle time and loss amount from a label.
 - Use a line controller and a system controller on the one line.
 - For the line controller
 - Connect the line controller to the stamping and engraving processors.
 - Use the OnJobStart trigger selection to set members label to the product ID label; note that the current value of the product ID being run can be obtained from the variables in the line controller tree using the command `getvarnum(current, "curprodtype") ;`

- Add the command `notifylinecontroller(..)` to the `onExit` trigger of the engraving processor.
- Create a simple schedule on the system controller and change over times on the line controller to times that can be checked easily.
- Once working, properly add downtime.
- Select all objects on the line, and use the duplicate function to create new lines.
- Use multiple sink objects to keep track of how many of each product is made.
- Create a global table production report for each line.
- Set up the system and line controllers.
 - On the line controller
 - Set the changeover times as specified in the exercise.
 - On the Product Data tab set the table to 7 products and 1 variable.
 - Populate the data as specified in the exercise.
 - Use the browse box to select the loss label on the processors.
- Change configuration of lines 1 and 2 with a single model.
 - Add a flag in the Operations table to indicate an old or new configuration.
 - Duplicate the processors on lines 1 and 2 to create a second processor for each line.
 - Add logic to the reset trigger of the processors that checks the configuration flag and sets the speed label accordingly; note that in the old configuration the input to the extra processor can be closed.
- Optional effort:
 - Use logic on the reset trigger of the system controller to transfer the appropriate schedule based on the configuration flag.
 - Use reset logic on the line controller to transfer the changeover data from the global table to the line controller changeover table.
- Visualization
 - Color the flowitems, starting at the stampers, using the product ID.

- Add a time of day using a visual tool

Figure A.15.1 provides a possible model layout for the exercise.

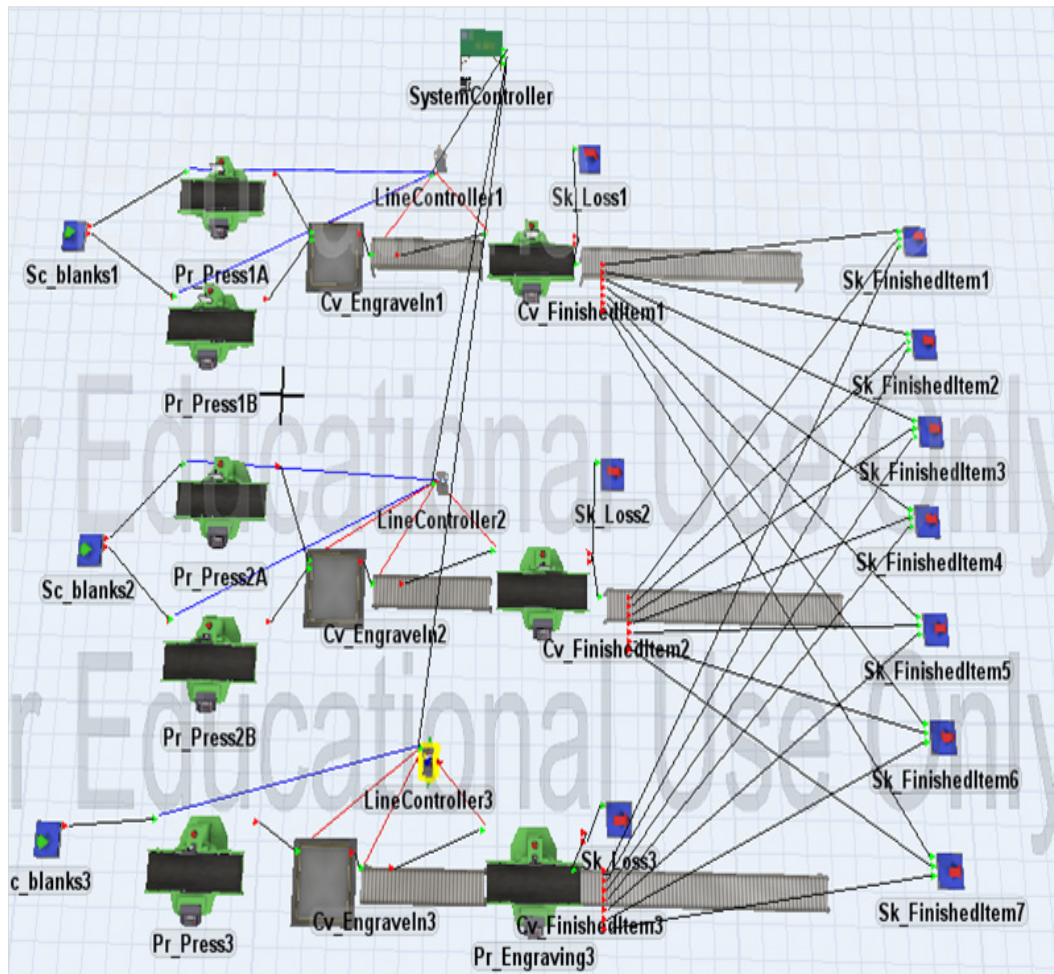


Figure A.15.1 Custom shapes simulation

Appendix - Interacting with Other Applications

This Appendix provides information on exchanging data between *FlexSim* and Microsoft *Excel* and importing AutoCAD drawings into *FlexSim*.

Section 1 – Microsoft Excel import/export

While the exercises in the text exchange input and output information between *FlexSim* and *Excel*, these actions are accomplished through a template format that does not need the user to write any scripting code. However, *FlexSim* has a full complement of commands for creating custom interfaces with *Excel* and other applications. These *Excel* commands are found in the Commands section under the Help menu and can be utilized by writing code in the simulation.

Data from global tables can be exchanged with *Excel* spreadsheets using normal Windows Cut and Paste commands; however, this process can be automated by using the interface shown in Figure A.I.1. It can be accessed via the *Excel* logo icon in the top row of *FlexSim* menu items or via the Tools>Excel drop-down menu. The interface options are described below:

- *Single Table Import*: Import a configured table into *FlexSim*. To configure the import table, press the Edit button to open the definition page. An example page is shown in Figure A.I.2.
- *Single Table Export*: Export a configured table from *FlexSim* to *Excel*. To configure the table for export, press the Edit button to open the definition page.

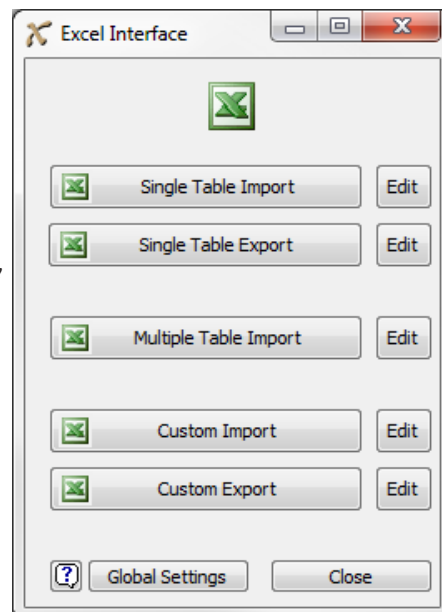


Figure A.I.1 Excel Import/Export window

- *Multiple Table Import*: Import several tables into *FlexSim*. Press the Edit button to open the definition interface page.
- *Custom Import*: Import from *Excel* using custom code. To write and edit this custom code, press the Edit button to bring up a code editor.
- *Custom Export*: Export to *Excel* using custom code. To write and edit this custom code, press the Edit button to bring up a code editor.
- *Global Settings*: Configure import/export parameters, including the install location of *Excel*, etc.

Figure A.I.2 shows the interface for defining how data for a global table can be imported into *FlexSim* from *Excel*. Basically, a path to an *Excel* workbook file and a spreadsheet page in that file is linked to a global table in *FlexSim*.

Commands for direct interaction with *Excel* are defined in the *FlexSim* Help/Commands drop-down menu, they can be embedded into logic scripts for execution during a simulation. Examples are included.

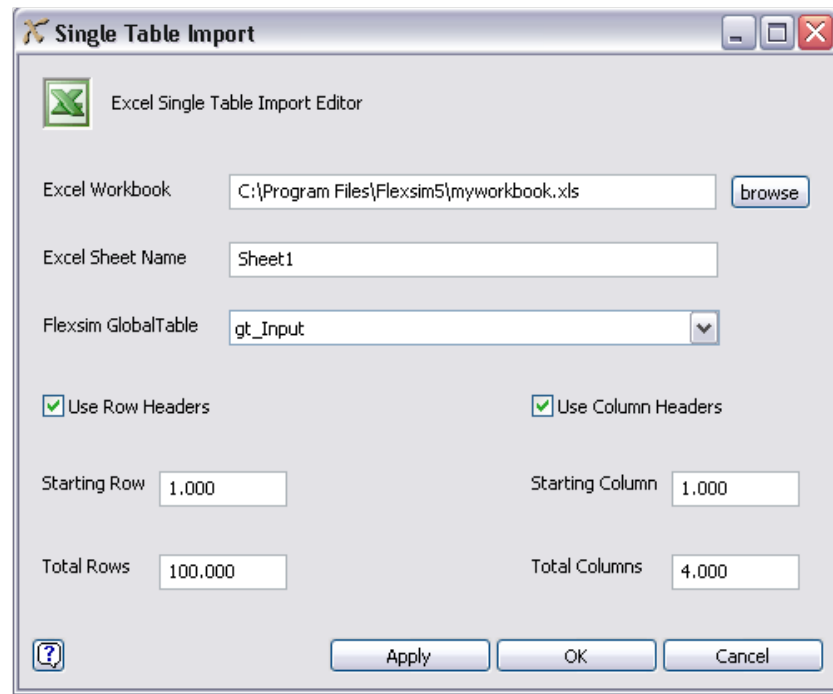


Figure A.I.2 Excel Table Import definition window

Section 2 - Importing AutoCAD drawings

The key to successfully importing *AutoCAD* drawings is to simplify overly complex drawings.

The following steps should be performed when preparing an *AutoCAD* .dxf file for import into *FlexSim*.

1. Remove all unnecessary information. *AutoCAD* files typically include much information that is unnecessary to the simulation. All a simulation usually

needs is a basic layout. Removing information that is extraneous to the simulation makes a model clearer and reduces the burden on the graphics card during a simulation. As a result, it is easier to build and present the model. Remove any parts of the drawing that are not pertinent to the simulation study.

2. Adjust the scale to *FlexSim* units. *AutoCAD* files are often scaled in inches. *FlexSim* models are often scaled in feet or meters. It is important that the *AutoCAD* file be rescaled to work appropriately in *FlexSim*.
 - For instance, to convert from an *AutoCAD* file that is in inches to a *FlexSim* model that is in feet, the scale factor will be 1/12. In general, to determine how much to scale, follow these steps:
 - Measure a known distance in AutoCAD (“_dist”).
 - Apply the following equation: scale factor = FlexSim distance / ACAD distance.
3. Scale the objects in *AutoCAD*
 - Select the objects to scale.
 - Type “_scale” in the command prompt or select the scale command from the menus.
 - Specify a reference point.
 - Type in the calculated scale factor in the command prompt.
4. Move objects to the origin. *AutoCAD* drawings are usually drawn using a specific coordinate system. This means that the objects usually are not located near the origin (0,0,0). When a .dxf file is imported into *FlexSim*, it is positioned in *FlexSim*’s coordinate system according to the .dxf’s positioning; therefore, if the origin point of the *AutoCAD* file is very far away from the actual drawing, when that .dxf file is imported into *FlexSim*, the layout will also be very far away from the model’s origin position in *FlexSim*. This can be quite frustrating for the modeler. For this reason, move the *AutoCAD* objects to the origin by completing the following steps within *AutoCAD*:
 - Select the objects to move.
 - Type “_move” in the command prompt or select the move command from the menu.
 - Specify a reference point.
 - Type in the desired location of that point in the command prompt.

5. Explode compound objects. *FlexSim* can only import basic shapes, so it is important to explode any compound objects in the AutoCAD drawing into their basic shapes. To explode compound objects in *AutoCAD* follow these steps:
 - Select the objects you want to explode.
 - Type “_explode” in the command prompt or select the explode command from the menus.
 - Repeat the above steps until all objects are exploded.
6. Continue within the *FlexSim* application:
 - Drag a VisualTool object into the model. Double click it to open its properties window.
 - Select “Imported Shape” for the Visual Display.
 - Select the *AutoCAD* file you want to import in the “Filename” field.

Appendix - Advanced Techniques

This Appendix covers a wide range of operational systems that can be modeled and analyzed using simulation and many of the features available in the *FlexSim* software; however, it doesn't cover the wide range of functionality that is available in *FlexSim*. This appendix summarizes some of those additional capabilities such as

- creating custom task sequences for transporters or other objects to follow;
- creating AVI files to document the simulation;
- developing custom user interfaces to control the simulations;
- using kinematics;
- generating reports, statistics, and documentation

More detailed explanations and examples are found in the *FlexSim* Help menu.

Section 1 Task Sequences

A task sequence is a series of tasks or actions that are executed in sequential order by a Task Executor object. Task Executors includes operators, transporters, cranes, ASRS vehicles, robots, elevators, and other mobile resource objects. Fixed resources, such as processors, have a default mechanism for creating task sequences in order to move flow items to the next station. They also automatically create a task for calling an operator during certain parts of the operation.

When the Use Transport field on a Flow tab of an object is checked, the following task sequence is created:

1. Travel to the object currently holding the item.
2. Load the item from that object.
3. Break.
4. Travel to the destination object.
5. Unload the item to the destination object.

When a Task Executer executes the above task sequence, it executes each task in order. Each task corresponds to a specific task type. Notice in the above example that there are two Travel task types in the task sequence, one Load task type, one Unload task type, and one Break task type; however, there are times when tasks are needed that have not been pre-defined.

Custom task sequences can be created using 3 simple commands:

```
createemptytasksequence()  
inserttask()  
dispatchtasksequence()
```

The steps for creating a custom task sequence are

- First, create a task sequence by using **createemptytasksequence()**.
- Then insert tasks into the task sequence by successive **inserttask()** commands.
- Finally, dispatch the task sequence with **dispatchtasksequence()**.

The following example instructs a forklift to travel to an object (referenced as “station”), then loads a flow item (referenced as “item”).

```
treenode ts = createemptytasksequence(forklift, 0,0);  
inserttask(ts, TASKTYPE_TRAVEL, station);  
inserttask(ts, TASKTYPE_LOAD, item, station, 2);  
dispatchtasksequence(ts);
```

In brief terms, **treenode** newtasksequence (ts in the example above) creates a reference, or pointer, to the task sequence as a *FlexSim* node so that it can be used later when tasks are added to the task sequence.

The **createemptytasksequence** command uses three parameters. The first parameter is the object that will handle the task sequence. This should be a Dispatcher or Task Executer object. The second and third parameters are numbers, specifying the task sequence’s priority and preempting values, respectively. The command returns a reference to the task sequence that was created.

The **inserttask** command inserts a task onto the end of the task sequence. Each task that is inserted has several parameters associated with it. The load task that was defined above will be used as an example.

There are two required parameters. The first is a reference to the task sequence into which the task is inserted; in this case, it is ts. The second parameter is the type of task; in this case, TASKTYPE_LOAD. This can be chosen from the list of tasks provided in the Users’ Manual or from the Help file.

The third and fourth parameters reference two involved objects, referred to as involved1 and involved2. These involved objects and what they mean depend upon the task type. For some task types, both involved parameters are needed and have meaning, whereas for others, the involved objects are not used. Some task types may use one involved object, and some have involved objects which are optional. Refer to the documentation for information on what a specific task type's involved objects represent. In the LOAD case, involved1 is what is to be loaded (in this case, item) and involved2 is from where (in this case, station).

The fifth, sixth, seventh, and eighth parameters are optional and define task variables var1-var4. By default, these values are zero. Again, their meaning can be found in the Help files. For the load task defined above, notice that var1 is specified as 2. For a load task, this specifies the output port through which the item will leave the station.

The **inserttask** command takes two or more parameters, which specify the task's values.

The User Manual in the Help file contains more detailed information for building task sequences that allow travel and logic to be customized to any situation.

Section 2 - Creating AVI files to document the simulation

The AVIMaker is available through the AVI Maker option in the Tools>Presentation menu. The basic interface is shown in Figure A.A.1. The object calls commands to make a video or AVI file of a model while it is running. The file will be made as long as it is in the model. If the user does not want to make the AVI file, they need to delete the object from the model.

Before running a model, designate a view to record. This is done by right-clicking on the desired view, right clicking, and selecting View | Designate this View (sv). The model may run very slowly while the AVI file is being created. It will not respond to the speed slider bar in the run control window during the record time.

AVI Maker Parameters

- *AVI Name:* Name of the file that the AVIMaker will write to. It should have an .avi extension.

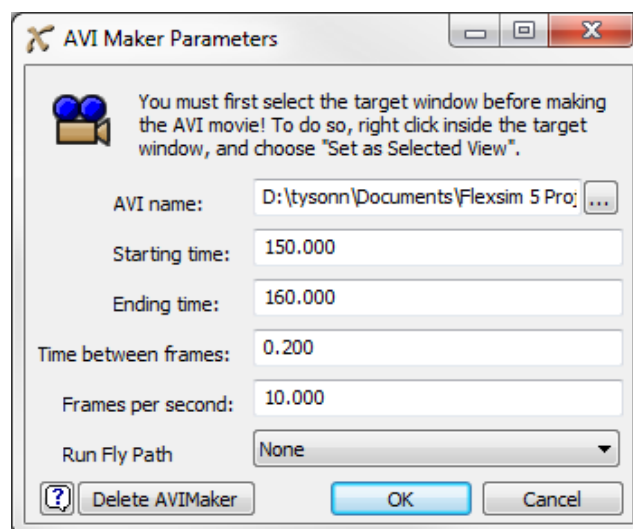


Figure A.A.1 AVI Setup Window

- *Starting Time*: Simulation time that the AVIMaker should start recording the model and begin creating the AVI file.
- *Ending Time*: Simulation time that the AVIMaker will stop recording and stop creating the AVI file. It is recommended that the model not be stopped before this time, as it may corrupt the avi file.
- *Time between frames*: How much simulation time passes in the model between recorded frames.
- *Frames per second*: Number of playback frames per second for the AVI file.
- *Run Fly Path*: If a fly path has been created in the model, it can be selected through this drop-down box. The view will follow that fly path as the model runs.
- *Delete AVIMaker*: When this button is pressed, the AVIMaker is deleted from the model. The model will run at normal speed again, and the AVI file will not be generated.

Additional details are provided in the User Manual located in the Help drop-down menu.

Section 3 - Creating custom GUIs

Graphical User Interfaces (GUIs) are accessed from the Tools menu. GUIs allow the creation of custom window interfaces to a model. They are especially useful when developing simulations for others to use.

FlexSim GUIs are made up of building blocks called views. Views are windows that can perform specialized roles and be combined hierarchically. Essentially, views allow data to be viewed and manipulated in the *FlexSim* tree structure. Since data takes many forms, there are many types of views.

This appendix creates a simple GUI to dynamically change the size of a queue as well as to report the output of the processor. To follow along with the discussion, create a simple model comprised of a source, conveyor, queue, processor, and sink, as shown Figure A.A.2. Using simple models is a good way to practice and understand how objects in a simulation work.

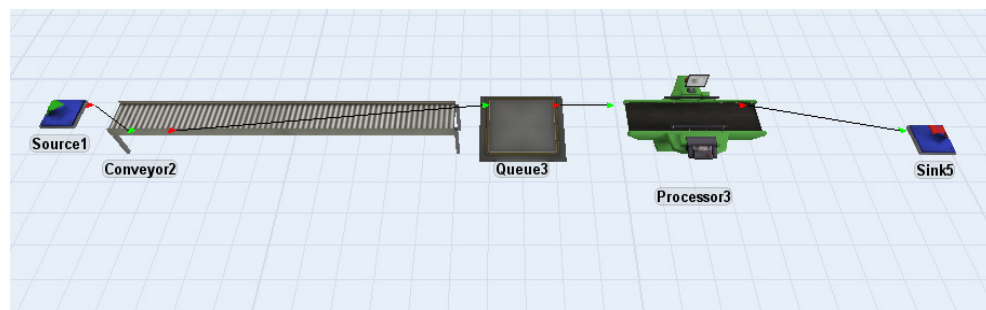


Figure A.A.2 Simple simulation for GUI example

To begin, select Add from the Graphical User Interfaces option on the Tools drop-down menu. As shown in Figure A.A.3, two windows will open. The window on the left is called the GUI Builder; it provides several tools for constructing custom GUIs. The window on the right is called the GUI canvas; it shows how the GUI will appear to the user. Initially it is blank, and GUI views are added to it in a drag-and-drop fashion from the GUI Builder window.

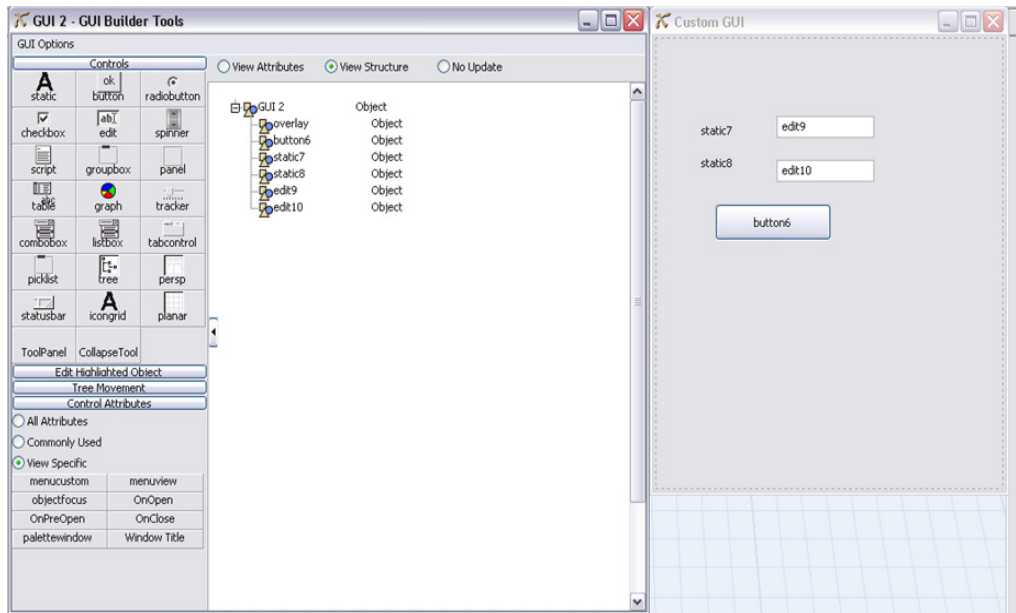


Figure A.A.3 Starting point for a Control GUI

Just as with simulation models, start by dragging out the items that are needed for the interface. From the top panel of the GUI Builder, drag a button onto the GUI canvas. A button should now appear on the GUI canvas where it was dropped.

Select the button by clicking on it. When the button is selected, a dotted outline appears around it, as well as a black square to the bottom right. To move the button, just click and drag it to a location. To change the size of the button, click and drag the black square. Next, add two static views and two edit views onto the GUI canvas.

Now change the names and attributes of the GUI views, as shown in Figure A.A.4. Click on the button. Then click on the View Attributes radio button in the GUI Builder window. Notice that the button and its attributes are

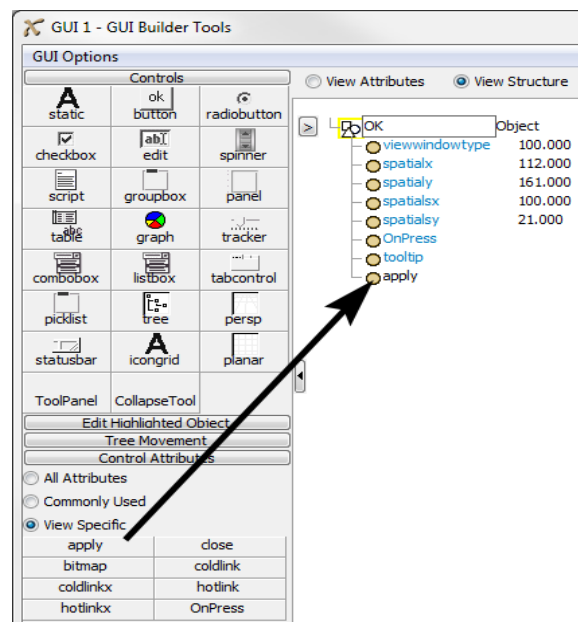


Figure A.A.4 changing a name and adding an attribute

now shown in the tree view of the GUI Builder. Click on the icon next to the word button in the tree view. A box appears around it, and the text can be changed.

For the name of the button, enter “OK” and then click outside the box to apply it. Toward the bottom, on the left of the GUI Builder window, in the Control Attributes section, there are some commonly used attributes that can be added to the button. To add an attribute, just drag it from the icon grid on the left to a blank area in the tree view.

Add an Apply attribute to the button. The Apply attribute causes all links found in the view to be applied to their respective destination nodes when the button is pressed.

Now click on one of the static label views. Again, the label and its attributes should appear in the tree view of the GUI Builder. For the name of the first label view, enter “MaxContent”. For the name of the second, enter “Output.”

To see the changes that have been made so far, refresh the GUI canvas to see the changes. Click on the GUI canvas window and then press the F5 button. This will change the GUI canvas from editing mode to regular viewing mode. Now the window should look like a normal window without the dotted lines around it.

Notice that the label views might not be large enough to fit the text. To fix this, go back into editing mode by selecting the GUI canvas window and pressing F5 again. Now rearrange the sizes and locations of the views so that there is enough room to show the entire text of the labels.

Finally, the edit views have to be connected to their proper nodes in the model. First, we’ll explain some basic concepts.

There are two types of links. Both types tell the view’s text field to be linked to a certain node in an object’s attribute tree:

- A *coldlink* attribute gets the value only once when the window opens, and it sets the value only when an apply button is pressed.
- A *hotlink*, on the other hand, continuously updates its text field as the value in the model changes.

The link text specifies a path to a node that is associated with the edit field. Since the GUI can link to any node anywhere in the simulation, different symbols are used in the path as quick ways of specifying how to traverse the tree to the destination node. They are very useful in creating complex relationships between the GUI and the simulation objects. Their meanings are as follows:

- @ Go to the owner view of the current node. For this coldlink node, it is the main GUI window.
- > Go into a node’s attribute tree.
- + Read the current node’s text as a path to an object, and go to that object.
- / Go into the current node’s sub-tree.
- .. Go to the current node’s parent tree, or up one level.

For the example shown in Figure A.A.5 only the > symbol is used. A coldlink is used to change the queue size, and a hot link is used to monitor the processor output.

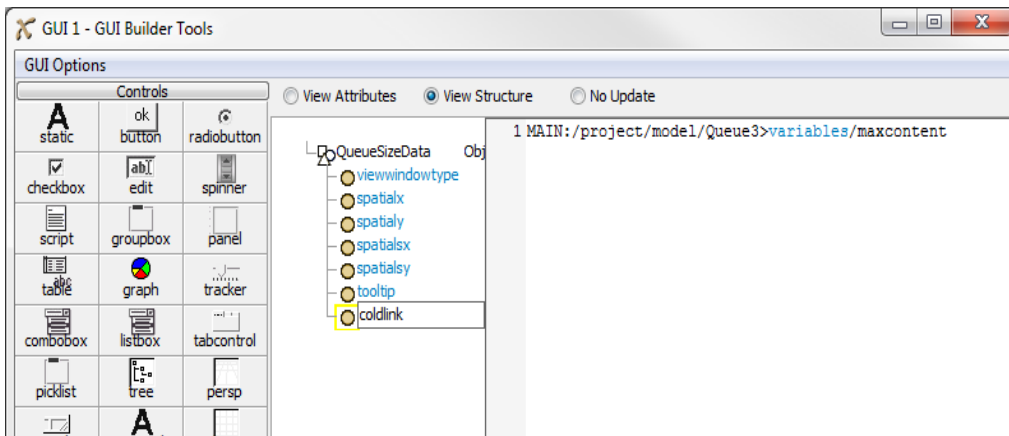


Figure A.A.5 Applying a Path

Click on the edit view for the queue size. Change the name and, from the attribute list, drag a coldlink to the tree. To set the correct path to the queue's maxcontent variable, click on the coldlink node icon in the tree. A window appears with the current text. Replace the text with the path. In this case, as shown in Figure A.A.5, a simple path will work and starts at the MAIN tree.

MAIN:/project/model/Queue4>variables/maxcontent

The coldlink path does the following:

- Starting at the very top of the simulation tree, a path is created to the queue object. Note that the object name has to be spelled correctly.
- From there, it goes into the object's attribute tree to find the node name variables (>variables).
- It then goes into the node's sub-tree and finds the node named maxcontent.

In a similar way, create a hotlink in the other edit view to reach the output of the processor. In the model, right click on the processor to look at its tree. It should be obvious that the correct path would be:

MAIN:/project/model/Processor3>stats/throughput/stats_output

One task remains: to identify the new GUI as the control GUI for the simulation. To do this, go to the top left corner of the GUI Builder window and click on GUI Options. From the list, select Assign this GUI to the "control" button. Close the GUI Builder using the X in the top right corner.

To open the control GUI, click on the button just to the left of the ? icon on the *FlexSim* menu bar. The GUI will appear as in Figure A.A.6. The GUI can be sized and positioned anywhere. Note that all of the control GUIs that were used in the exercises for Section 3 were developed using the GUI Builder.

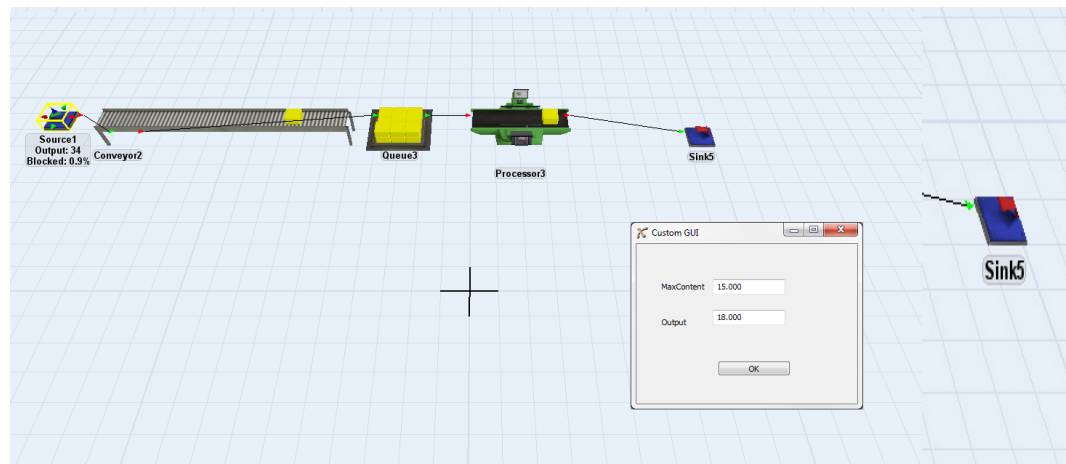


Figure A.A.6 GUI on the simulation surface

Section 4 - Kinematics

Kinematic functionality allows a single object to perform several animated movements simultaneously, each travel operation having its own acceleration, deceleration, start speed, end speed, and maximum speed properties. For example, an overhead crane usually has several motors that drive it. One motor drives the bridge along a railing, another motor drives a trolley along the bridge, yet another lifts the hook or grabber by a cable. Each of these motors may have their own acceleration, deceleration, and maximum speed properties. Having these different motors work simultaneously gives the motion of the crane a very dynamic behavior and look.

Before kinematics were introduced, the simplest way to simulate this behavior was to have three different objects hierarchically ordered in the model tree, each object simulating one motion or kinematic. This, however, could often become tedious and unfriendly. Kinematic functionality fixes this problem by allowing one object to do several motions or kinematics simultaneously.

To perform kinematic operations, the user must first call the **initkinematics** command. This initializes data for the kinematics, saving things like the start location and rotation of the object that you want to apply motion to. Once the kinematics are initialized, the subsequent travel/rotate operations to the object are given using the **addkinematic** command.

For example, an object can be told to start in 5 seconds and travel 10 units in the x direction, with a given acceleration, deceleration, and maximum speed. Then the object can be told to travel 10 units in the y direction, starting in 7 seconds; different acceleration, deceleration, and maximum speed values can be specified. The effect of these two operations is that the object starts traveling in the x, then starts simultaneously accelerating in the y direction, thus following a parabolic curved path to the destination. Each of the individual operations is added using the **addkinematic** command. The view is refreshed during the kinematic motion by calling **updatekinematics**; it calculates the current position and rotation of the object.

For the **initkinematics** command, as well as all other kinematics commands, a reference to a blank node must be passed as the first parameter. This specifies where

the kinematic information is stored. The node needs to be an otherwise unused node, so that kinematic functionality can store data as needed. For modelers, this node will most likely be a label. Once kinematics have been initialized, you can click on the node to view information for the kinematics. See the *FlexSim* User Manual for complete details.

Section 5 - Reports, statistics, and documentation

FlexSim has facilities for generating various types of reports and charts. To enable data collection and reporting, Full History must be enabled before the simulation is started. The history capability is set by selecting Full History On in the Statistics drop-down menu that is located on the bar at the top of the *FlexSim* window.

With Full History On, data are recorded during a model's run and saved to a database file. This database file is cleared on reset, unless saved. Once the run has completed, and before the model is reset, go to the Statistics drop-down menu on the tool bar at the top of the screen and select Reports and Statistics. This produces the tabbed report window shown in Figure A.A.7.

The tabs (Summary Report, State Report, and Model Documentation) correspond to the standard *FlexSim* reports. By clicking on a tab, selecting the material to be reported, and selecting the Generate Report button in the bottom right corner, the report is opened in *Excel* as a .csv file.

The Options tab allows the user to specify classes of objects to be included in the report.

FlexSim Chart is a tool that can be used to enhance the analysis of a model run. The results of the simulation are displayed in customizable graphs and charts that can be exported for use in other reports. Use of *FlexSim Chart* starts after a run is completed by selecting the Full Report tab of the Reports and Statistics window. A database is created from the data collected during the run. To create the database, click on the Generate Report button at the lower right hand corner of the Full Report tab. Select an older file to write over or name a new one.

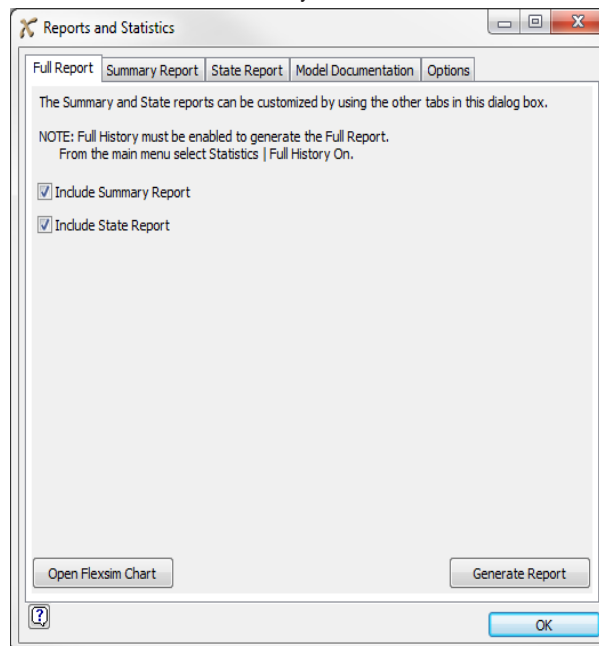


Figure A.A.7 FlexSim Report window

The data records that are created consist of two tables, which are deleted when the model is reset. Those tables are

- Movement information – when a flowitem moves
 - Time
 - Source object
 - Target object
 - Flowitem's unique ID
 - Itemtype
- State changes – when an object changes state
 - Time
 - Object
 - New state

In addition to these main tables, the database contains a list of objects, lists of groups, as well as the standard overall summary and state reports. The data file is saved as a Microsoft *Access* .mdb file. *Access* does not have to be installed for *FlexSim Chart* to operate and the user need not interact with the file directly.

FlexSim Chart is opened by clicking on the button in the lower left corner of the Reports and Statistics Full Report tab window. When asked for a file to open, navigate to the one that was just generated or to another. As a result, a *FlexSim Chart* window, like the one in Figure A.A.8, opens.

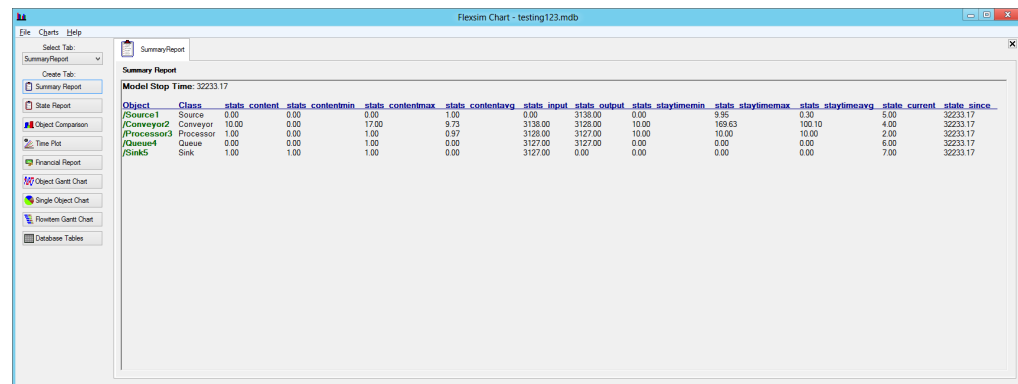


Figure AA.8 *FlexSim Chart* Window

Reports are created by clicking on the appropriate Create tab. The Summary and State reports in *FlexSim Chart* are the same as those reports generated by the Report tab of the Reports and Statistics window. The other reports require specifying the chart attributes.

An example of the window for the Object Comparison chart is shown in Figure A.A.9.

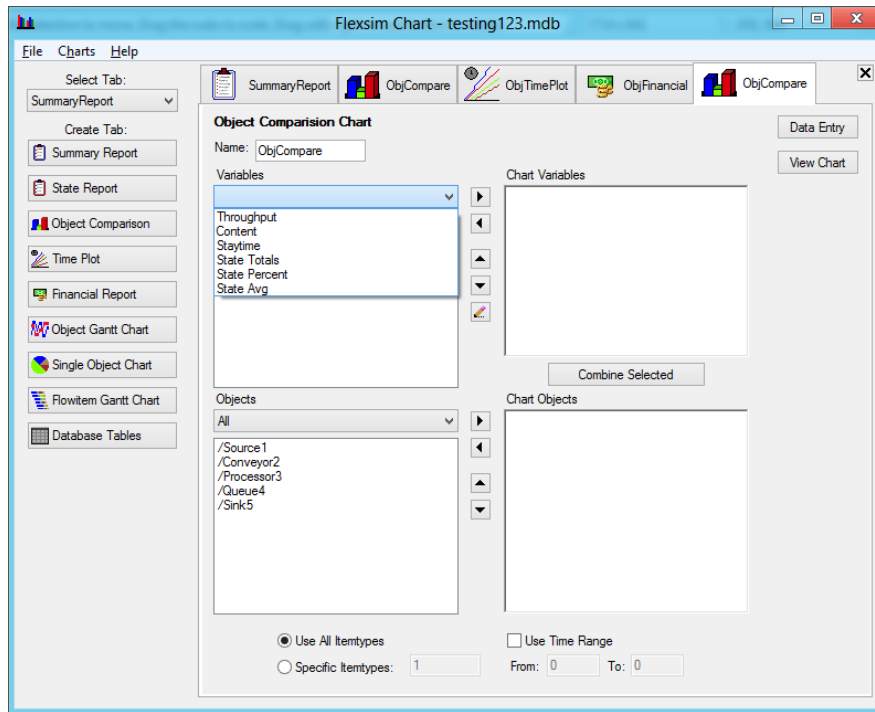


Figure A.A.9 *FlexSim Chart* Object Comparison Window

Choices for the Variable and Object frames can be seen by clicking on their drop-down arrow. Selected choices are moved to the final chart areas using the arrows between the window frames. When finished, select View Chart on the upper right side of the window.

Charts created in this way are shown as tabs in the tool bar at the top. Each tab defines one report. Nine types of tabs can be created and multiple instances of each tab type can be created. Tabs can be closed or just hidden.

These *FlexSim Chart* sessions can be saved and re-opened at any time.

Appendix - Related Applications

The robustness of the *FlexSim* software architecture has allowed it to be the basis for other applications. These focused applications, which are based on the same engine, include *FlexSim Healthcare*, *FlexSim CT*, and *Express It 3D*.

Section 1 - Health Care

Medical facilities are oftentimes extremely complex. Numerous factors contribute to their overall efficiency and effectiveness of work-flow, including

- patient flow (as shown in Figure A.R.1);
- staff utilization;
- resource management (as shown in Figure A.R.2).

FlexSim Healthcare was developed in association with healthcare professionals to create a tool for modeling the complexities and nuances of healthcare management—all without the need for programming.

The application is a 3D simulation tool. Users can build and interact with the simulation directly in the 3D environment. It enables a better understanding of the medical system, its interactions, and how to improve it.

FlexSim Healthcare includes a library of objects that were developed specifically to facilitate building models of any healthcare system. These objects can be customized using drop-down options.

FlexSim Healthcare has been used to analyze management and staffing requirements. Facility simulation allows hospitals and clinics to adopt lean methods while still providing

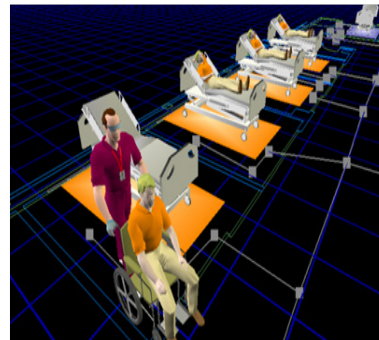


Figure A.R.1 Transporting a patient

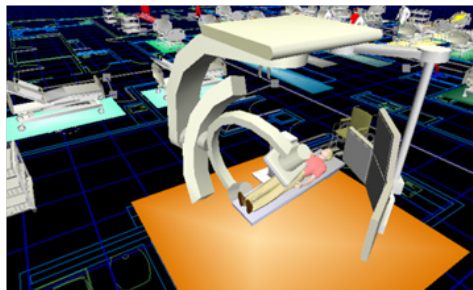


Figure A.R.2 Healthcare equipment

improved health care to their patients. Results have shown that a productive staff makes for more satisfied patients.

Section 2 - Container Terminal (CT)

The world-wide economy has created the need for improving the efficiency of port operations, especially in container ship loading and unloading. *FlexSim CT* was created using the base *FlexSim* engine, but creating an environment of modeling and analysis tools specific to the container port industry. Simulation models built in *FlexSim CT*, such as the one shown in Figure A.R.3, are used to improve many facets of container terminal operations. Specifically, the areas of improvement include the following:

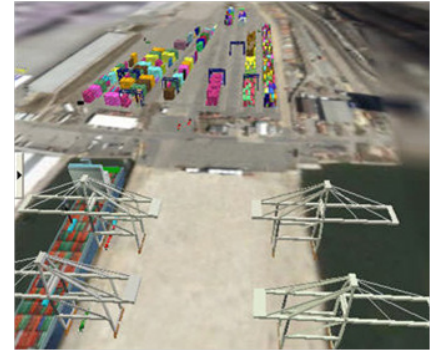


Figure A.R.3 Container Terminal

- Increased throughput
- Improved equipment utilization
- Reduced waiting time and queue sizes
- Reduced bottlenecks
- Balanced workload and allocating resources efficiently
- Analysis of alternative investment ideas
- Demonstration of new tool design capabilities
- Operator training in overall system behavior and job related performance

The *FlexSim CT* application has also been used to improve such operational issues as the following:

- Yard stacking and segregation strategies
- Yard layout and dispatching
- Gate logistics and capacity
- Quay crane allocation
- Yard equipment allocation
- Hostler gang allocation
- Ship scheduling
- Berth assignment
- Traffic constraints

FlexSim CT has helped find ways to move more containers more efficiently and at less cost. Ideas for improving a process can be simulated, tested, and justified.

As with any other simulation, *FlexSim CT* allows end-users to confidently communicate their assumptions, findings, conclusions, and future actions to colleagues, managers, or clients about how to improve container flow specifically and terminal operations generally. In addition to demonstrating improvements, organizations and individuals using simulation gain a greater insight into the system being studied, which consequently helps managers to predict and control a terminal's operations with greater reliability in the future.

FlexSim CT was the first commercially available off-the-shelf simulation tool for container terminals. As shown in Figure A.R.4, the simulation model provides real-time statistics on many key performance measures, including the following:

- Berth productivity
- Ship queuing and wait times
- Crane utilization
- Truck queuing and wait times
- Yard content and container dwell times
- Yard equipment utilization
- Gate throughput and turn times

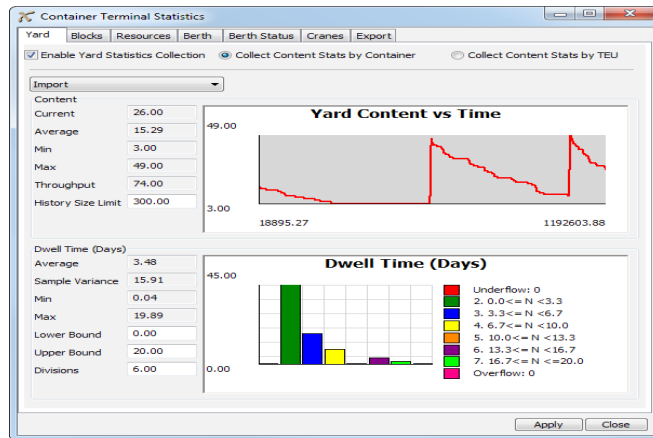


Figure A.R.4 CT simulation output

Section 3 - Express It 3D presentation software

Based on the 3D visualization of *FlexSim*, *Express It 3D* was built from the ground up to take advantage of that capability for use in presentations.

Express It 3D also takes advantage of the *FlexSim* tree structure and drag-and-drop model building methods. The user selects 3D content from a library (called inventory in *Express It 3D*) adds it via a drag-and-drop mechanism. The *Express It 3D* inventory contains 3D objects, 3D text, 3D animated objects, video objects, and light source objects.

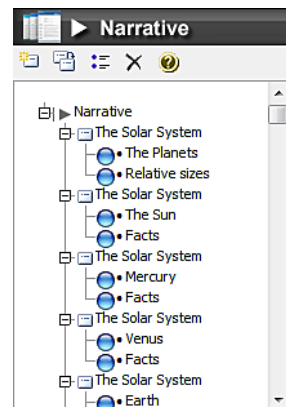


Figure A.R.5 Presentation tree structure

A presentation in *Express It 3D* is called a *narrative*. A narrative is organized in a tree structure, as shown in Figure A.R.5. The items in the tree view represent slide pages and points in the narrative. This structure makes organizing a presentation very easy.

The narrative allows re-arranging of pages and bullet points by dragging them to a new order. The text on the page becomes the header text and the point text becomes the text in the presentation. *Express It 3D* includes a spelling checker that checks all of the text in the presentation.

As shown in Figure A.R.6, slides can contain a virtual world canvas. The virtual worlds are built with objects from the inventory. In the virtual world, spatial relationships can be created by integrating custom 3D shapes and multimedia. Fly paths can be built to move about the world environment. These worlds can then be embedded into points on a page.

Because *Express It 3D* uses *OpenGL*, realistic spatial relationships, dynamic lighting, and environmental effects can be created. *Windows Media Player* is embedded into the application allowing a full range use of video. Specifically, video clips (as AVI files) can be used as background for slides, included as points on a slide, or shown as continuously animated texture on a 3D shape.

Express It 3D can import *PowerPoint*[®] presentations and convert them into a 3D presentation automatically. *Express It 3D* also provides a complete user guide, as well as several roadmaps that are context sensitive to help the user along the way.



Figure A.R.6 Presentation created by *Express It 3D*

About the Authors

Malcolm Beaverstock, Ph.D. After a 40 year career applying advanced control and simulation to industrial problems, Malcolm Beaverstock retired to join FlexSim. During his 13 years with General Mills, he developed and led their simulation program which was involved with more than 300 projects and resulted in significant savings attributed to the use of simulation.



Following graduate studies, Malcolm joined UniRoyal Chemical as manager of their advanced control group where he installed direct digital control systems. During his 17 years at the Foxboro Company he managed various technology groups and established the systems research department that resulted in the early development of Foxboro's I/A control system. As Vice President of Automation Technology, Malcolm initiated work into model-based control applications.

Malcolm holds a Bachelor's degree in Chemical Engineering and Labor Relations from MIT and a Doctorate from Cornell University in Chemical Engineering and Computer Science. He's the author of more than 200 papers on the application and use of advanced technologies.

Allen G. Greenwood, Ph.D., P.E. As Professor of Industrial and Systems Engineering at Mississippi State University (MSU) Allen teaches systems simulation, enterprise systems engineering, and project management. His research interests/expertise include the design and analysis of production and project systems; simulation modeling, analysis, and optimization; and the design and application of decision-support systems.



His professional experience spans a wide variety of domains -- engineering design and development (military aircraft and aerospace), manufacturing and production systems (military aircraft, shipbuilding, automotive, textile fibers, healthcare, and electrical systems), and project management. His work has been funded by such organizations as the Air Force Research Laboratory, Office of Naval Research, Naval Sea Systems Command, NASA, Northrop Grumman Ship Systems, Nissan North America, General Electric Aviation, and Center for Advanced Vehicular Systems (MSU). He has authored or co-authored over 100 creative works, including journal and conference papers, technical reports, software programs, etc.

Allen received his B.S.I.E, M.S.I.E, and Ph.D. (Management Science) degrees from North Carolina State University, University of Tennessee, and Virginia Tech, respectively. Prior

to joining MSU, he held positions at American Enka Company, General Dynamics Corporation, Virginia Tech, Northeastern University, and the American University of Armenia.

Eamonn Lavery, Ph.D. Eamonn completed his first object-oriented simulation software while in graduate school. In 1999 he was hired by F&H Simulations B.V. to conduct research and development for new simulation products. He developed the OpenGL virtual reality animation views for Taylor ED simulation software. In 2003 Eamonn joined FlexSim Software Products, Inc. as the Chief Technology Officer. He is the author and architect of the FlexSim Software.



Eamonn holds a Bachelor of Science in Mechanical Engineering and a Doctorate in Object Oriented Modeling and Simulation of Manufacturing Systems from Queens University of Belfast.

William Nordgren, MS CIM. Bill is the President and CEO of FlexSim Software Products, Inc. In 1988 he founded ProModel Corporation and was Vice President until he left in 1992. In 1993 Bill founded F&H Simulations, Inc. (now FlexSim Software Products, Inc.) and introduced Taylor II, Taylor ED, and FlexSim into the market.



Bill has authored several papers dealing with simulation project management, queuing theory, and has taught hundreds of classes in the use of simulation software. He is listed in Marquis Who's Who in America for his accomplishments in the advancement of simulation technology. Bill received a Bachelor of Science in Manufacturing Engineering Technology, and a Master of Science in CIM (Computer Integrated Manufacturing) from Brigham Young University.

Index

A

Advanced User 14, 16, 17, 18, 166, 225, 227, 260, 313
AGV 77, 149
Analyst 68, 105, 165, 244
Arrival sequence 25, 48, 55, 110, 335
Arrival style 335
ASRS 150, 395
Assumption 23, 39, 67, 76, 79, 95, 198, 206
Attribute 89, 112, 114, 115, 120, 131, 132
Availability 30, 34, 72, 207, 209

B

Batch 25, 27, 28, 29, 34, 58, 111, 145, 272, 274, 275, 276, 277
Bottleneck 12, 13, 26, 32, 56, 158, 209, 248, 276
Boundaries 79
Boundary assumptions 79, 95
Breakdowns 5, 25, 46, 85, 89, 100, 206, 215, 222, 360, 361, 372
Breakdown tab 114, 212, 213, 214
By expression 122, 123, 134, 137, 328, 331

C

C++ 233, 234, 367, 370
CAD 81, 96
Centerobject 237, 363, 373
Center port 16, 116, 119, 154, 215, 231, 336, 337, 363
Changeover 81, 287, 387, 388
ChangeOver Table 287
Cleanout Time 287
Code edit 313, 392
Code template 239, 312
Combiner 111, 143, 144, 145, 146, 149, 160, 161, 214, 215, 256, 257, 271, 335, 360, 361, 378
Competing failure 210, 211, 215
Complexity 4, 25, 27, 29, 32, 72, 76
Conceptual model 67, 68, 73, 75, 318
Connections 81, 114, 116, 117, 118, 119, 120, 122, 136, 154, 214, 237, 285, 287, 310
Container 8, 83, 97, 111, 144, 145, 146, 155, 178,

217, 236, 237, 272, 366, 408, 409
Content 85, 120, 122, 123, 154, 206, 238, 240, 244, 248, 272, 343, 361, 365, 377, 380, 409
Controller 17, 253, 254, 283, 284, 285, 286, 287, 288, 289, 293
Conveyor 80, 87, 111, 112, 114, 119, 120, 121, 122, 153, 229, 255, 284, 309, 313, 314, 315, 316, 336
 Conveyor layout 122, 313
 Conveyor object 119, 315
Copy From Highlighted 310
Creation time 378, 381
Current object 134, 135, 137, 237, 240, 244, 245, 248
Custom code 206, 230, 233, 239, 241, 283, 324, 372, 392
Custom logic 16, 17, 121, 136, 227, 233, 236, 238, 244, 260, 275, 286
Cycle time 35, 109, 126, 165, 228, 248, 254, 288, 371, 372, 377, 379, 387

D

Data engine 368, 369
Data nodes 229, 230
Decision support 5, 6, 19, 48, 65, 68, 70, 71, 73
Decision Support 71
Decision variables 47, 49, 124, 187, 189, 191, 202, 317
Degree of complexity 25, 32
Dempirical 165, 332, 347, 380
Designer 68
Design of experiments 91, 367
Developer 68, 90, 92, 131, 169
Discrete empirical 165, 178, 333, 347
Discrete event 23, 33, 34, 35, 269, 273, 281
Discrete object 109, 110, 149, 272, 273, 281
Dispatcher 153, 215, 337, 396
Domain knowledge 69
Down function 215, 216, 361
Downtimes 30, 31, 55, 80, 85, 91, 137, 156, 208, 209, 210, 211, 212, 213, 214, 215, 216, 341, 388
Drop-down menu 105, 108, 109, 113, 118, 122, 134, 135, 213, 215, 242, 271, 298, 310, 316, 324, 358, 391, 392, 398, 399, 403

E

Editing objects 114
Educator 69
Efficiency 3, 6, 12, 28, 56, 60, 94, 95, 205, 292, 299, 361, 374, 407, 408
Errors 12, 202, 233, 239, 240, 241, 242, 243, 290, 330, 346
Experimenter 47, 187, 196, 197, 203, 355, 357, 358, 360, 361, 373
ExpertFit 68, 165, 166, 167, 168, 175, 345, 346

F

Failure clock 207, 210, 211, 212, 214, 215, 217
Failure mode 32, 210, 211, 215, 217
FIFO 158, 176, 177, 180, 182, 183, 189, 199, 200, 201, 237
File menu 297
First available 119, 135, 136, 271
Fixed resources 84, 85, 86, 100, 110, 112, 118, 120, 122, 129, 143, 150, 319, 340, 395
Flag 25, 372, 377, 380, 381, 384, 385, 388
Flexscript 230, 233, 234, 235, 238, 242, 244, 245, 247, 248, 324, 363, 370
Flexscript commands 230, 235, 238, 244, 245, 247, 248, 363
Flowitem 109, 110, 119, 121, 122, 132, 143, 144, 145, 146, 254, 272, 323
Flow logic 126, 136, 146, 320, 331
Flow node 120, 122, 128, 129, 153, 316, 379, 380, 381
Fluid 269–281
Fluid blender 271, 281
Fluid flow 35, 269, 270
Fluid Generator 271
Fluid mixer 272, 274, 281
Fluid objects 109, 270, 271, 272, 273, 274, 280, 281, 320, 383
Fluid pipe 271
Fluid Processor 272
Fluid splitter 272
Fluid tank 271, 272, 386
Fluid terminator 271
Fluid to item 272
Functional specification 45, 77, 79, 81, 90, 91, 92, 93

G

General tab 115, 116, 120, 155, 258, 259, 311, 312, 315, 316
Global table 139, 165, 213, 245, 246, 247, 248, 256, 257, 320, 332, 333, 347, 348, 359, 360, 383
Global variable 242, 243, 245, 246
Graphical editor 156, 341, 342
Grid 108, 109, 112, 120, 151, 152, 153, 271, 300, 310, 316, 319, 338, 400
GUI 70, 71, 228, 298, 300, 398, 399, 400, 401, 402

H

Health care 9, 31, 407, 408
Highlighting 139, 233, 310
HighMark 383, 384, 385

I

Ideal rate 205
Implementer 69
Input port 118, 119, 144, 161, 237, 248, 254, 285, 287, 342, 363, 377, 378
Interface control 297, 300
Intermediate User 15, 16, 18, 103, 105, 143, 227, 233, 256, 313
Investigator 69
Involved object 339, 397
Item to fluid 272
Itemtype 132, 133, 135, 136, 142, 183, 238, 325, 326, 329, 330, 331, 332, 343, 360, 364, 377, 380, 404

J

Job shop 155, 203, 340
Join 30, 55, 88, 111, 145, 317, 318

L

Labels 18, 112, 114, 131, 132, 133, 230, 238, 256, 310, 319, 323, 328, 365, 371, 376, 377, 378, 380, 381, 400
Label-table 256, 257, 261, 263, 376, 378
Language engine 368, 370
Lean 2, 3, 11, 17, 18, 25, 26, 30, 32, 38, 39, 41, 71, 216, 217, 275, 277, 407
Level indicator 273
Library 17, 18, 108, 112, 113, 149, 155, 251, 260, 261, 271, 369, 370, 385, 407, 409

Library icon grid 108, 271
Line controller 283, 284, 285, 286, 287, 288, 289, 293, 387, 388
Logic 23, 67, 80, 82, 89, 114, 131–142, 146, 153, 227, 230, 238, 240, 244, 255, 275, 331, 388

M

Main screen 107, 108
Main toolbars 108, 113
Manufacturing cell 203, 252, 261
Material distribution 30
Maximum rate 205
Members 213, 214, 215, 287
Message 83, 85, 132, 133, 137, 240, 243, 251, 252, 253, 254, 366, 377, 386
Message trigger 132, 366
Methodology 60, 61, 64, 75, 76, 82, 87, 118, 125, 163, 169, 188, 202, 322
Metrics 23, 25, 29, 37, 45, 46, 49, 51, 52, 79, 95, 127, 138, 141, 205, 206, 209, 263, 280, 383
Midmark 383, 384, 385
Mission length 207
Mixer recipe 274
Mixer steps 274
Mobile resources 84, 85, 100, 110, 122, 135, 143, 149, 161, 395
Modeling 1, 21, 22, 23, 25, 47, 63, 65, 66, 67, 68, 72, 81, 86, 105, 106, 110, 198
Modeling paradox 40
Modeling utilities 261, 309, 310
MTBF 85, 207, 208, 209, 211, 212, 213, 214, 215, 217, 219, 248
MTTR XI, 85, 207, 208, 209, 211, 212, 213, 214, 215, 217, 219, 248
MultiProcessor 111

N

Naming convention 114, 140
Network layer 367, 370
Network node 149, 153, 154, 158, 337, 338, 339, 343, 360, 380, 386
No Connect 338, 339
Node 120, 122, 128, 129, 149, 153, 154, 158, 228, 229, 230, 231, 234, 235, 236, 237, 238, 243, 247, 257, 288, 316, 337, 338, 339, 343, 360, 363, 365, 367, 375, 376, 379, 380, 381, 386, 396, 400, 401, 402, 403

Data node 229, 230
Flow node 120, 122, 128, 129, 153, 316, 379, 380, 381
Tree node 230, 243
Variables node 230, 238

No Passing 338, 339

O

Object attributes 364
Object control 366
Objective 25, 61
Object labels 365
Object nodes 229, 230, 236, 237, 238, 247
Object oriented 34
Object spatial attributes 364
Object statistics 15, 123, 124
Object variables 274, 367
Occasional User 13, 14, 15, 16, 19, 43, 45, 46, 48, 60, 61, 76, 92, 297
OFD 14, 15, 82, 83, 86, 87, 100, 125, 126, 138, 148, 158, 160, 263, 277, 280, 292, 307, 318
OnCreation trigger 232, 325, 326, 332
OnEntry trigger 133, 137, 182, 231, 236, 245, 247, 257, 259, 288, 324
OnExit trigger 133, 137, 182, 232, 238, 288, 320, 325, 326, 329, 359, 376
OnReset trigger 133, 243, 246, 253, 288, 371
OpenGL 367, 370, 410, 412
Operability 39, 90
Operating assumptions 79
Operations system 4, 5, 19, 24, 26, 38, 40, 41, 61, 63, 120, 143, 160, 161, 185
Output port 118, 119, 149, 232, 237, 272, 273, 327, 331, 363, 397

P

Pack 145
Passing 153, 338, 339
Performance 5, 12, 13, 408
Performance measure 14, 16, 17, 23, 37, 47, 78, 85, 87, 91, 92, 101, 123, 165, 177, 181, 188, 189, 190, 191, 192, 193, 194, 195, 197, 199, 202, 205, 206, 266, 278, 305, 317, 318, 322, 358, 361, 409
Picklist 115, 134, 136, 144, 146, 154, 233, 238, 275, 311, 312, 313, 324, 326, 332, 333, 335, 375
Picklist options 115, 134, 144, 146, 233, 238, 312, 313, 326, 332, 333
P&ID 81

Planning rate 205
Port rate 272
Preempt 215, 216, 337
Priority 40, 58, 140, 141, 150, 156, 157, 158, 215, 216, 248, 319, 333, 337, 342, 396
Probabilistic 124, 163
Probability distributions 23, 46, 68, 163, 165, 166, 168, 169, 172, 173, 175, 176, 178, 179, 181, 182, 347, 348
Procedure 63, 80, 82, 90, 95, 97, 133, 159, 178, 260, 264, 265, 266
Processor 111, 134, 231
Product Data 286, 287, 388
Production line 26, 30, 37, 79, 92, 269, 283, 284, 286, 287
Production schedule 14, 25, 29, 60, 95, 283, 284, 285, 293
Programmed model 67, 73
Project template 77, 90, 92, 93, 94, 98, 127, 316
Properties window 115, 131, 394
Pseudocode 245, 247
Pull 3, 31, 32, 84, 135, 137, 142, 289, 381
Pull production 289

Q

Queue object 110, 119, 325, 401
Queuing theory 23, 24, 87, 192, 412

R

Recorder 251, 257, 375, 376, 378
Referencing objects 236, 363
Reliability 17, 27, 32, 60, 72, 91, 114, 206, 207, 209, 210, 213, 214, 215, 283, 291, 409
Repair time 96, 165, 185, 207, 208, 209, 210, 211
Researcher 69
Reset 49, 113, 129, 133, 194, 245, 246, 297, 298, 299, 312, 339, 372, 377, 380, 383, 385, 388, 403
Resources 319, 320
Round robin 136, 142
Run Time 58, 79, 108, 129, 297

S

Screen elements 154
Selecting 49, 51, 70, 116, 136, 165, 239, 243, 256, 260, 299, 310, 312, 323, 336, 337, 357, 397, 400, 403

Senddelayedmessage 251, 252, 366
Sendmessage 251, 252, 366
Send to Port 135, 136, 142, 232
Separator 111, 146, 161, 272
Service operations 1, 8, 106
Set itemtype 332
Simple cell 86
Simulation benefit categories 10
Simulation control panel 108
Simulation engine 369
Simulation knowledge 69
Simulation languages 33, 34, 251
Simulators 7, 22, 33, 34, 35, 46, 83
Sink 110, 122, 212, 217, 229, 245, 246, 330, 360, 378, 380, 381, 388, 398
SMA life cycle 65
SMA process 66, 75
SMA project 65, 68, 69
Source 71, 83, 110, 219, 232, 271, 404
Space 109
Speed control 298, 373, 374
Speed Limit 338
SQL 367
Staffing 3, 13, 28, 39, 48, 50, 51, 52, 96, 264, 407
Stakeholders 11, 45, 64, 65, 67, 69, 72, 73, 75
State 37, 71, 123, 403, 404, 411
Statistics tab 49, 117, 123, 257, 316
Staytime 123, 206
Stochastic 22, 46, 47, 61, 124, 163
Stop time 51, 108, 244, 298
Subnodes 229, 230, 236, 376
Surge 26, 32, 216
Switch 233, 240, 241, 252, 378
System controller 17, 283, 284, 285, 286, 387, 388

T

Target 144, 205, 206, 248, 288, 346
Target rate 205
Task executers 112, 122, 129, 143, 149, 150, 153, 215, 216, 320, 335, 336, 337, 339, 380
Throughput 13, 48, 58, 78, 87, 94, 117, 128, 146, 182, 199, 200, 201, 203, 217, 322, 361, 401, 408, 409
Ticker 270
Time controls 108
TIS 245

Tools tab 155

Tree 15, 18, 228, 229, 230, 234, 235, 236, 237, 243, 257, 288, 357, 375, 376, 387, 398, 400, 401, 402, 409, 410

Tree node 231

Triggers 114, 132–142, 182, 231, 253, 256, 273, 310, 358, 383
 Flow trigger 134, 142, 328, 331, 372, 381

Trigger tab 132, 142, 273

Troubleshooting 243

U

Unpack 146, 161

User 7, 13, 14, 15, 16, 17, 18, 19, 43, 45, 46, 47, 48, 60, 61, 76, 89, 92, 103, 105, 138, 143, 150, 155, 166, 225, 227, 233, 242, 256, 257, 260, 271, 297, 298, 313, 321, 340, 346, 368, 370, 375, 376, 397, 398, 399, 403

User interface 89, 370

User interface engine 370

User library 251, 260, 261

Use transport 362

Utilization 11, 12, 23, 56, 78, 85, 87, 93, 128, 138, 141, 203, 322, 331, 333, 407, 408, 409

V

Validation 32, 39, 40, 73, 90, 91

Value stream map 14, 81, 82

View Settings 309

Virtual distance 153, 343, 360, 380

Visual Display 154, 311, 394

Visualization 120, 143, 151, 152, 154, 227, 228, 258, 278, 367, 370, 381, 409

Visual tool 154, 155, 245, 246, 247, 253, 254, 261, 311, 381, 389

VR 368, 370

W

Wizards 105

Workstation 33, 86, 87